UNIVERSIDAD DE CUENCA

FACULTAD DE INGENIERÍA

Escuela de Ingeniería Electrónica y Telecomunicaciones

"DESARROLLO DE UN VEHÍCULO AUTÓNOMO TERRESTRE PARA NAVEGACIÓN ÓPTIMA EN AMBIENTES CERRADOS"

Trabajo de Titulación previo a la obtención del título de Ingeniero en Electrónica y Telecomunicaciones

Autores:

Danilo Abraham Calle Sarmiento
C.I. 0105987911

Christian David Sosoranga Maldonado
C.I. 0105802516

Director:

Dr. Luis Ismael Minchala Ávila
C.I. 0301453486

Cuenca - Ecuador
2017

Resumen

Actualmente, la Robótica constituye un campo activo de investigación que busca desarrollar prototipos parcial o completamente autónomos, integrando inteligencia artificial, visión artificial, algoritmos de evasión de obstáculos y algoritmos de planificación de rutas. En este contexto, el presente trabajo de titulación expone la implementación de un robot autónomo, usando el robot DaNI basado en una tarjeta sbRIO-9631. La programación del prototipo es realizado mediante la plataforma LabVIEW que incluye módulos y herramientas usadas en la Robótica y procesamiento digital de imagenes (DIP): LabVIEW Robotics, LabVIEW FPGA, LabVIEW Real-Time, LabVIEW Vision Development, etc. Los algoritmos de planificación de rutas se implementan en Matlab. Las principales técnicas estudiadas en este documento son los métodos basados en muestreo y los métodos basados en espacios. Los métodos basados en muestreo están representados por los algoritmos Rapidly Exploring Random Tree (RRT) y Bidirectional Exploring Random Tree ($B - RRT$); a su vez, los métodos basados en espacio están presentados por los algoritmos A star (A^*), Fast Marching Method (FMM) y Fast Marching Square (FMM^2). Finalmente, el robot autónomo DaNI es simulado en entornos 3D virtuales y luego se comprueba su funcionamiento en entornos reales.

Palabras clave: UGV, DaNI, Atificial Vision, RRT, B-RRT, A Star, FMM, FMM^2.

Danilo Calle
Christian Sosoranga M.

Abstract

Nowadays, robotics is an active research field devoted to the development of partially or complettely autononomous vehicles, integrating artificial intelligence, artificial vision, obstacle evasion algorithms and path planning algorithms. This research prsents details on the implementation of an autonomous vehicle, using the DaNI robot based on sbRIO-9631 card. The programming plataform is LabVIEW, which includes modules and tools for robotics and digital image processing (DIP): LabVIEW Robotics, LabVIEW FPGA, LabVIEW Real-Time, LabVIEW Vision Development, etc. The path planning algorithms are implemented In Matlab. The main techniques are studied in this document are the sampling-based, and space-based methods. The sample-based methods are represented by the Rapidly Exploring Random Tree (RRT), and Bidirectional Exploring Random Tree ($B-RRT$) algorithms; on the other hand, the space-based methods are represented by the A star (A^*), Fast Marching Method (FMM) and Fast Marching Square (FMM^2) algorithms. Finally, the autonomous DaNI robot is simulated on 3D virtual environments, and also its operation is evaluated in real environments.

Keywords: UGV, DaNI, Atificial Vision, RRT, B-RRT, A Star, FMM, FMM^2.

Danilo Calle
Christian Sosoranga M.

Índice general

Índice de figuras

Índice de tablas

Yo, *Danilo Abraham Calle Sarmiento*, autor del Trabajo de Titulación *"Desarrollo de un Vehículo Autónomo Terrestre para Navegación Óptima en Ambientes Cerrados"*, certifico que todas las ideas, opiniones, y contenidos expuestos en la presente investigación, son de exclusiva responsabilidad de sus autores.

Cuenca, 15 de mayo de 2017

Danilo Abraham Calle Sarmiento
C.I. 0105987911

Yo, *Christian David Sosoranga Maldonado*, autor del Trabajo de Titulación *"Desarrollo de un Vehículo Autónomo Terrestre para Navegación Óptima en Ambientes Cerrados"*, certifico que todas las ideas, opiniones, y contenidos expuestos en la presente investigación, son de exclusiva responsabilidad de sus autores.

Cuenca, 15 de mayo de 2017

Christian David Sosoranga Maldonado
C.I. 0105802516

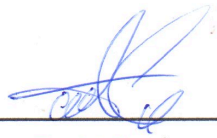
Yo, *Danilo Abraham Calle Sarmiento*, autor del Trabajo de Titulación *"Desarrollo de un Vehículo Autónomo Terrestre para Navegación Óptima en Ambientes Cerrados"*, reconozco y acepto el derecho de la Universidad de Cuenca, en base al Art. 5 literal c) de su Reglamento de Propiedad Intelectual, de publicar este trabajo por cualquier medio conocido o por conocer, al ser este requisito para la obtención de mi título de *Ingeniero en Electrónica y Telecomunicaciones.* El uso que la Universidad de Cuenca hiciere de este trabajo, no implicará afección alguna de mis derechos morales o patrimoniales como autor.

Cuenca, 15 de mayo de 2017

Danilo Abraham Calle Sarmiento
C.I. 0105987911

Yo, *Christian David Sosoranga Maldonado*, autor del Trabajo de Titulación *"Desarrollo de un Vehículo Autónomo Terrestre para Navegación Óptima en Ambientes Cerrados"*, reconozco y acepto el derecho de la Universidad de Cuenca, en base al Art. 5 literal c) de su Reglamento de Propiedad Intelectual, de publicar este trabajo por cualquier medio conocido o por conocer, al ser este requisito para la obtención de mi título de *Ingeniero en Electrónica y Telecomunicaciones.* El uso que la Universidad de Cuenca hiciere de este trabajo, no implicará afección alguna de mis derechos morales o patrimoniales como autor.

Cuenca, 15 de mayo de 2017



Christian David Sosoranga Maldonado
C.I. 0105802516

Agradecimientos

Aunque solo un par de personas aparezcan en la portada, son muchas las que contribuyeron para que este merecimiento hoy sea una realidad, seguramente me olvidaré de algunos; espero que me perdonen y me lo recriminen para no olvidarme en una próxima ocasión.

Antes que nada, quiero agradecer a Dios por proveerme de salud y sabiduría para afrontar cada uno de los retos habidos y por haber. Agradezco a la vida por permitirme disfrutar la compañía y cariño de mi madre, padre y hermanos. A mis tíos Julio Sarmiento, Román Sarmiento y Elsa Sarmiento, y a mis abuelitos por apoyarme emocionalmente y económicamente, alentándome cuando me quería bajar del tren. A mis dos tíos que cambiaron mi forma de ver la vida, Amable Calle y Celso Calle, gracias por haberme permitido formar parte de sus vidas colmadas de grandes experiencias, siempre los llevaré en el corazón. Y por ultimo, a mi Familia les agradezco por ejemplificar mi vida: vivir bien, osadamente, sin reproches, pero sobre todo, con humildad. Hoy, saber que ustedes me aconsejaron es un privilegio, pero saber que yo lo predique es un lujo.

Agradezco al Dr. Ismael Minchala por darme la oportunidad de trabajar con él, quién más allá de su preparación académica también ha demostrado una calidad humana semejante. Gracias por la paciencia y aguante mostrado. Finalmente, a mis amigos por la ayuda brindada a lo largo de esta pequeña etapa de mi vida, fue un honor haberlos conocido, espero que vivan sus vidas osadamente, que se exijan, que no se conformen y que sean colmados de bendiciones. Que Dios les bendiga!

Danilo Calle.

Agradecimientos

A todas las personas que me han ayudado, enseñado y guiado de forma desinteresada e incondicional. A las personas que supieron tolerarme y me han agasajado con su presencia y compañía, gracias. Padres, hermanos, amigos, profesores, compañeros y más, por el echo existir, gracias.
Por último, gracias amada e inolvidable U. de Cuenca.

Christian David Sosoranga Maldonado.

Dedicatoria

Este trabajo de titulación se lo dedico a Dios por darme salud, fuerza y sabiduría, enseñándome a nunca desfallecer ante las adversidades, sin perder la fe, la dignidad y el orgullo de realizar las cosas con rectitud.

A mi familia por el apoyo incondicional en aquellos momentos difíciles. A mi madre Narcisa Sarmiento y hermana Tania Calle que constantemente se preocupan de mi bienestar, insistiendo siempre con un plato de comida cuando mis ocupaciones me retraían más de lo habitual. A mis tíos Amable Calle y Celso Calle quienes se convirtieron en mi motivación e inspiración; siempre los llevaré en mi memoria. A mis amigos que hicieron de esta lucha más placentera, por esas experiencias vividas y esa amistad forjada. Al Doctor Ismael Minchala por su paciencia y por los conocimientos transmitidos. Gracias a todos.

Danilo Calle.

.

A quienes en los más profundo de su ser creen y confían en mí. También a quienes en calma y profundo silencio esperan que cambie.

A quienes aún le sonríen a la vida...

Christian Sosoranga M.

Capítulo 1

Introducción

1.1. Antecedentes

La definición de robot ha trascendido más allá del significado acuñado por Karel Capek en 1921, en su obra de teatro *RossumsUniversalRobots* (RUR, por sus siglas en ingles), y se ha convertido en el término que define uno de los más grandes temas de investigación y desarrollo en la actualidad, la Robótica [18]. Generalmente, el término robot es asignado a maquinas creadas por el hombre para su propio beneficio. Los robots abarcan aplicaciones en el sector de la automatización industrial, la medicina, milicia, entretenimiento, etc. En la industria, los robots son empleados en la ejecución y supervisión de tareas repetitivas, potenciando el correcto uso de recursos y garantizando un trabajo de calidad. En la medicina, los robots son usados, entre otras aplicaciones, en asistencia a pacientes en rehabilitación de lesiones que limitan el movimiento, incluyendo asistencia en locomoción a través de exoesqueletos. Por otra parte, en la milicia, los robots son usados esencialmente en tareas de reconocimiento como búsqueda de bombas, exploración de terrenos inhóspitos; previniendo la pérdida de vidas humanas [19]. La generación masiva de robots converge a un conjunto de prioridades de procesos de automatización como: generar cierto nivel de autonomía y navegación [20]. Con este propósito surgen los vehículos autónomos (UV, por sus siglas en inglés).

Los UV constituyen un dominio de investigación donde varias disciplinas se juntan, como la inteligencia artificial, aprendizaje de máquinas y visión por computadora [21]. Diversos tipos de UV que se han desarrollado en virtud de su especialidad. Entre los UV más reconocidos se encuentran: vehículos autónomos terrestres (UGV, por sus siglas en inglés), vehículos aéreos autónomos (UAV, por sus siglas en inglés), vehículos autónomos submarinos (UUV, por sus siglas en inglés) y vehículos autónomos para el mar (USV, por sus siglas en inglés) [22].

Este trabajo está enfocado específicamente a los UGV que se desenvuelven en ambientes cerrados (interiores), con condiciones de iluminación controladas.

Los UGV son vehículos complejos que necesitan la conjuncion de varios sistemas para ser eficientes [22, 23]. Los problemas que enfrentan los UGV son:

- Planeación de trayectorias
- Detección y evasión de obstáculos
- Problemas de navegación
- Control
- Autonomía

Ya que los UGV priman en trabajos de exploración y mapeo de zonas de alto riesgo, el motor principal de los UGV son los algoritmos de rastreo, exploración, monitoreo, visión artificial, e identificación y evasión de obstáculos. El complemento de estos algoritmos involucra el uso de diversos sensores y actuadores, con la finalidad que el robot obtenga las características del entorno en el cual transita.

Este trabajo de investigación involucra el desarrollo de un UGV utilizando la plataforma NI LabVIEW Robotics Starter Kit (**DaNI**) para entornos cerrados no estructurados o parcialmente estructurados mediante la integración de recursos tanto de *hardware* como *software*. El robot DaNI es capaz de desplazarse de forma autónoma desde un punto de inicio a un punto meta, determinando la presencia de obstáculos y redirigiendo su ruta de ser necesario.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar un vehículo autónomo terrestre a través de la integración de hardware y software que permita navegación autónoma en ambientes cerrados.

1.2.2. Objetivos específicos

1. Implementar características sensoriales de un robot móvil desde la problemática del posicionamiento

2. Implementar algoritmos de navegación para evadir obstáculos empleando sensores de ultrasonido, luz estructurada y visión artificial

3. Analizar e implementar algoritmos para generar trayectorias y seleccionar la ruta óptima

4. Integrar los algoritmos de navegación que brinden autonomía al robot DaNI

1.3. Contribuciones del trabajo de titulación

Los aportes de este trabajo de titulación son los siguientes:

- Integración los recursos necesarios de *hardware* y *software* al robot DaNI, con el fin de convertirlo en un UGV
- Algoritmos de generación y optimización de trayectorias en ambientes cerrados no estructurados o parcialmente estructurados para navegación autónoma del robot del robot DaNI
- Metodología de integración de recursos para manejo adecuado del consumo de energía durante la navegación en una trayectoria planificada

Capítulo 2

Fundamentos teóricos

2.1. Robótica móvil UGV

Los vehículos no tripulados se basan en la toma de decisiones propias para realizar misiones; por lo tanto, no existe la necesidad de la actuación o respaldo de una persona, pues éstos vehículos incorporan en su lógica algoritmos de planificación de rutas. Los UGV actualmente están en auge, existen varios prototipos consolidados (robots *Spirit Rover* y *Opportunity Rover* [4], vehículos *Stanley* y *Boss* [5]), todo gracias al trabajo de investigación realizado durante décadas.

2.1.1. Evolución

La evolución de la robótica móvil ha presentado un crecimiento exponencial gracias a la inteligencia artificial. **Shakey** fue el primer robot en utilizar inteligencia artificial para controlar sus movimientos en 1968 [1]. Asimismo, en aquel entonces, apareció el problema de deslizamiento. Fenómeno producido por el cálculo aproximado de la posición actual, usando información de la posición previa, velocidad y aceleración. Por tal motivo, el robot producía un error acumulativo al desplazarse una cierta distancia, ocasionando un conocimiento errado de su posición. La Figura 2.1 muestra al robot Shakey.

Otro robot destacado fúe el **Stanford Cart**, desarrollado en la Universidad de Stanford a finales de los setenta. Éste se basaba en el seguimiento de una trayectoria definida por una línea en la superficie [2]. La Figura 2.2 muestra al Stanford Cart.

Asimismo, en la década de los setenta, empieza el desarrollo de plataformas de

Figura 2.1: Robot Shakey [1]

Figura 2.2: Standford Cart [2]

exploración por parte de la administración nacional de la aeronáutica y del espacio (NASA, por sus siglas en inglés) y el laboratorio de propulsión a chorro (JPL, por sus siglas en inglés). El primer robot elaborado fúe el **MARS-ROVER**. Este robot poseía las características del robot SHAKEY, pero el modelo era diferente. Por lo que, el mapa resultante consistía de polígonos irregulares que representan las áreas donde puede trasladarse [1]. El siguiente robot construido por JPL fue el **SOJOURNER**. Éste descendería en Marte el 5 de julio de 1997 y tres meses después transmitió fotografías, tomas de temperatura y análisis químicos de la superficie [3]. La Figura 2.3 muestra al SOJOUNER.

Figura 2.3: Sojouner [3]

Desde enero de 2004 los robots teleoperados **Spirit Rover** y **Opportunity Rover** se encuentran funcionando en Marte. Tienen implementados laboratorios

geológicos móviles, con diversas herramientas de estudio de la superficie. La dimensión de estos robots gemelos es de 1.6 m con un peso de 174 Kg. Por su parte, el Sojouner tiene un tamaño de 65 cm y pesa 10 Kg. El Sojouner viajó 150 m en la superficie de Marte, comunicado con una base fija, mientras que el Spirit y Opportunity tienen varios kilómetros recorridos y continúan con la exploración. El sistema de comunicación de los robots gemelos lo llevan incorporado cada uno de ellos. Los robots Spirit y Opportunity llevan incorporado un software para navegar y eludir obstáculos. Esto les permite trazar su trayectoria, ya sea desde la tierra o por autonomía. Para ello, los robots tenían una velocidad de 5 cm/seg. en terreno plano, cuando son operados desde la tierra y en modo automático posee una velocidad de 1 cm/seg. El software incorporado se fundamenta en el uso de imágenes estereo. El robot recepta las imágenes y genera un mapa tridimensional del entorno, calculando la densidad, altura de las rocas y rugosidad del terreno, con el fin de determinar si el terreno es transitable. Al momento de finalizar el cálculo, como resultado entrega varias rutas posibles. En consecuencia, el robot decide una ruta considerando la seguridad y la distancia al punto de destino predefinido. Por último, los robots recorren entre medio a dos metros y recalculan la ruta trazada [4]. La Figura 2.4 se ilustra al robot Spirit Rover.

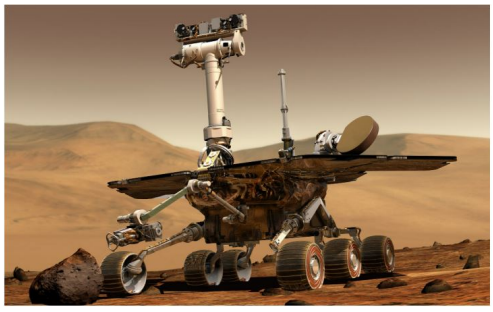


Figura 2.4: Robot Spirit Rover [4]

En USA durante el 2004 y 2005 la *Defense Advanced Research Projects Agency, USA*(DARPA, por sus siglas en inglés) desarrolló una competencia para vehículos, los cuales debían atravesar el desierto de Mojave en California recorriendo más de 200km en forma autónoma, siguiendo un mapa de coordenadas GPS y transitando por terracería; los vehículos paricipantes fueron construidos por universidades como Stanford y Carnegie Mellon, y por empresas como Ensco y Velodyne Acoustics. En 2007, se realizó otra competencia, pero en un ambiente urbano, lo cual representó un reto mayor. Los vehículos ganadores fueron **Stanley** (Figura 2.5) desarrollado por la Universidad de Stanford y **Boss** (Figura 2.6) desarrollado por

el *Tartan Racing Team*, el cual estaba conformado por estudiantes y personal e investigadores de diferentes entidades de Carnegie Mellon University, General Motors, Caterpillar, Continental e Intel [5].

Figura 2.5: Vehículo Autónomo Stanley [5]

Figura 2.6: Vehículo Autónomo Boss [5]

Los vehículos Stanley y Boss constan de sensores láser para ubicar los obstáculos del alrededor, cámaras RGB para ubicar el camino, unidades de medición inercial (IMU, por su siglas en inglés) con un sistema de posicionamiento global (GPS, por su siglas en inglés) para localizar el robot respecto al recorrido que debe seguir. La desventaja de estos vehículos es que dependen de ubicación GPS y de un costoso sensor 3D. Por ello, la universidad de Oxford se encuentra desarrollado desde el 2015 el vehículo **RobotCar UK** que usa láseres y cámaras mucho más baratas, y se localiza mediante información visual-espacial del entorno (no requiere GPS) [5].

Finalmente, con el ánimo de seguir incentivando la investigación relacionada a este campo, diversas empresas han desarrollado plataformas robóticas de bajo costo, donde los estudiantes puedan implementar desde tareas simples, como

el control de un motor utilizando el algoritmo proporcional-integral-derivativo (PID), hasta algoritmos complejos, que incluyen localización y navegación óptima. Un ejemplo palpable son las empresas National Instruments (NI, por sus siglas en inglés) y Pitsco con su prototipo el robot **DaNI**, que provee la oportunidad de aprender conceptos de ingeniería a través de aplicaciones prácticas.

El robot DaNI es empleado en la educación, en desarrollo de aplicaciones de percepción y control con el software LabVIEW. Consta de un chasis, tren de transmisión, ruedas, motores, transductores, una tarjeta única reconfigurable E/S NI 9631 sbRIO y cableado (para mayor información sobre el robot DaNI y sus características, revisar el datasheet [1]). Una de sus características más importantes, es que su hardware puede ser estudiado, modificado e inclusive aplicar ingeniería inversa. Sin embargo, su enfoque principal es la percepción del robot y los fundamentos de control que se implementan en el software LabVIEW desarrollado en un *host* remoto [24]. La Figura 2.7 muestra al robot DaNI e interacción de sus componentes principales mediante diagrama de bloques.

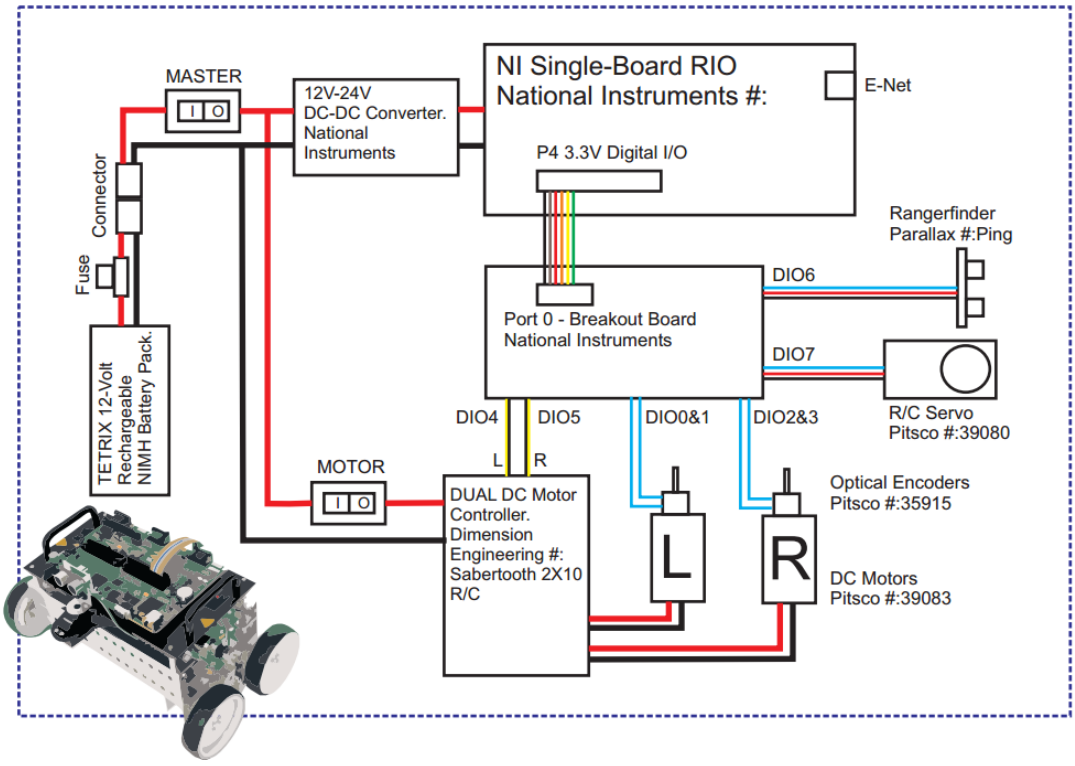


Figura 2.7: Diagrama de bloques de DaNI [6]

[1] Datasheet del robot DaNI: http://www.ni.com/datasheet/pdf/en/ds-217

2.1.2. Aplicaciones de los UGV

De acuerdo al trabajo que desempeñan los UGVs, existen una amplia gama de aplicaciones derivadas. Sin embargo, a consideración nuestra, se rescatan dos de las más importantes: UGVs en la minería y UGVs en ambientes hostiles.

UGVs en la minería

En la minería, la necesidad de transportar grandes cantidades de material de forma segura y eficiente, sugiere el uso de robots con características diferentes al de un robot convencional. Por lo tanto, estos robots deben contar con actuadores de gran potencia y sistemas autónomos, con el fin de optimizar la producción y seguridad.

El estado del arte de la robótica en este ámbito tiene los siguientes avances:

- Robots autónomos equipados con sistemas de perforación y toma de muestras guiados por GPS. La Figura 2.8 se muestra un prototipo de vehículo autónomo perforador de la marca Atlas Copco.

Figura 2.8: Vehículo autónomo perforador [7]

- Robots equipados con sistemas de tunelación, capaces de encontrar bolsas de gas en el subsuelo
- Robots colocadores de explosivos para voladuras automáticas y seguras
- Camiones autónomos para carga de material. La Figura 2.9 se muestra un prototipo de vehículo autónomo perforador de la marca Atlas Copco.

Figura 2.9: Camiones con sistema autónomo de carga a la mina West Angelas, Australia Occidental [8]

UGVs en ambientes hostiles

En ambientes hostiles (temperaturas extremas, radiación, contaminación extrema), donde la seguridad de las personas se ve comprometida, se requiere el uso de personal calificado y equipos de protección costosos. Generalmente, éstos entornos son de difícil acceso, ya sea por su tamaño u orografía. Por tanto, la robótica es la mejor opción para interactuar con estos entornos.

Los robots más utilizados en ambientes hostiles son los siguientes:

- El robot desactivador de minas. Estos robots se basan en el uso de sensores especiales en detección de artefactos. Mientras que, otros realizan el barrido de un área específica detonando los explosivos ocultos con herramientas diseñadas especialmente para estos dispositivos. La Figura 2.10 muestra el prototipo de un robot desactivador de minas creado por la agencia estatal intermediaria para la exportación e importación de productos relacionados con la defensa Rosoboronexport S.A.

- Una aplicación similar a la anterior es el robot desactivador de bombas. La finalidad de la construcción de este robot, es la neutralización de bombas colocadas por terroristas o criminales en lugares con alta afluencia de personas. Al mismo tiempo, estos desactivadores protegen la integridad de las personas. Por el hecho de que, no se envía un escuadrón antibomba, sino que solamente un robot desactivador de bombas. La Figura 2.11 ilustra el prototipo de un robot desactivador de bombas creado por DARPA.

Figura 2.10: Robot Urán-6 desactivador de minas [9]

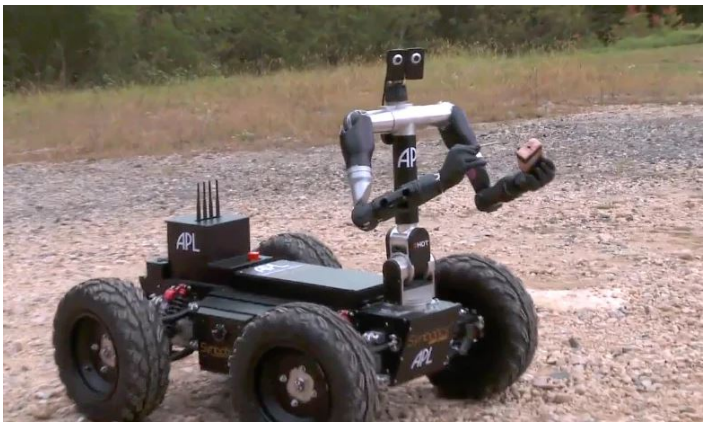


Figura 2.11: Robot Robo Sally desactivador de bombas [10]

- El prototipo de la Figura 2.12 es un robot autónomo que mide, monitorea y mapea rayos gama en terreno al aire libre. Para ello, el sistema de medición de rayos gama se adaptó a un sistema robótico Orpheus-X4 para que se pueda desplazar en los diferentes terrenos. Este robot posee algoritmos para navegar de forma autónoma en un área predefinida basados en el sistema global de navegación por satélite (GNSS, por sus siglas en inglés) [11].

  El UGV tiene la finalidad de realizar mapas de radiación, adquiridos en el área de seguridad durante los experimentos. Estos mapas son presentados con una alta precisión de la posición y se comparan con datos de posición del GPS.

Figura 2.12: Robot Orpheus-X4 [11]

2.2. Visión artificial

La visión artificial o visión por computador es un campo de la ingeniería situado dentro de la inteligencia artificial, la cual trata el estudio de imágenes digitales incluyendo métodos de adquisición, procesamiento, análisis y extracción de información de imágenes, que permite capturar información visual de un entorno físico para extraer características visuales relevantes.

Otro criterio para la visión artificial es el permitir detectar automáticamente la estructura y propiedades de un entorno (dinámico o estático) tridimensional a partir una o más imágenes bidimensionales. Estas imágenes pueden ser cromáticas o monocromáticas, las que pueden ser capturadas por una o más cámaras estacionarias o móviles [25]. La estructura y propiedades deducidas de un entorno por medio de la visión artificial permiten el análisis de propiedades geométricas (tamaños, formas, localización de objetos, entre otras), propiedades del material (sus colores, sus texturas, la composición y otras) y la luminosidad de las superficies.

Una definicion formal de la visión artificial es la siguiente: *«Ciencia que estudia la interpretación de imágenes mediante computadores digitales»* [25].

La visión artificial es aplicada en varias disciplinas, por tanto, no existe una formulación concreta y estándar de la problemática de la visión artificial y menos del cómo resolver los problemas con visión artificial.

Por otra parte, existe una gran cantidad de métodos para resolver tareas bien

definidas, en dónde la metodología es generalmente muy específica y difícilmente puede ser generalizada para un amplio rango de aplicaciones. Varios métodos y aplicaciones de la visión artificial aún se encuentran en una básica investigación, pero muchos otros han logrado hacerse productos comerciales formando en muchos casos parte de un sistema mayor capaz de resolver problemas complejos.

2.2.1. Objetivo de la visión artificial

El objetivo general es obtener una descripción de una escena, dotar a las máquinas de un sistema de percepción visual similar al de los humanos, para lo cual se realiza un análisis de las regiones de interés, extrayendo rasgos particulares para luego ser analizados e interpretados. Generalmente los datos de las regiones de interés se obtienen mediante el procesamiento de imágenes digitales (DIP, por sus siglas en inglés).

En el caso específico del presente trabajo de titulación, la visión artificial se centrará en al adquisición y procesamiento de imágenes digitales haciendo uso del software de desarrollo National Instruments LabVIEW y sus diferentes módulos dedicados para la visión artificial (NI Vision), facilitando la adquisición y procesamiento de imágenes digitales.

2.2.2. Componentes del sistema de visión artificial

Un sistema de visión artificial se compone de varios recursos de hardware, software y algoritmos (recursos teóricos). Para su desarrollo y análisis el sistema suele ser dividido en cinco elementos principales o sub sistemas, los cuales se enuncian a continuación.

Elementos principales de un sistema de visión artificial [26]:

1. Iluminación
2. Captación
3. Adquisición
4. Procesamiento
5. Periféricos

Cada uno de los elementos enunciados anteriormente se pueden apreciar en la Figura 2.13. Estos tienen crucial importancia para un óptimo resultado del sistema completo. A continuación, se comenzará tratando el concepto de imagen digital y posteriormente se hará una breve referencia a cada uno de los elementos del sistema de visión artificial. En el Capítulo 3, es caso de ser necesario, se dará una mayor atención a los elementos particulares que se usen en el desarrollo del proyecto.

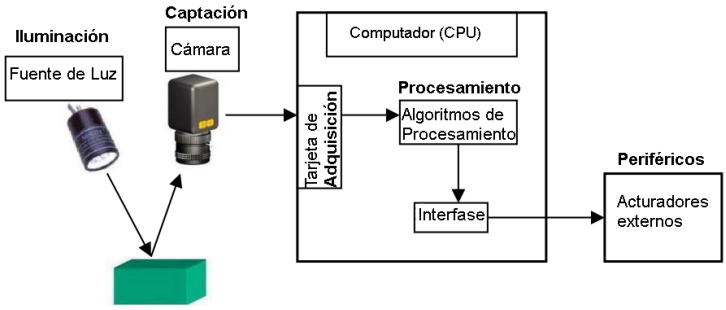


Figura 2.13: Elementos del sistema de visión artificial [11]

Imagen digital

Una imagen digital es un arreglo bidimensional de valores que representan la intensidad de luz. Esta se representa como una función de la intensidad de luz (2.1), donde f es el brillo o luminosidad del punto *(x,y)*, y x y y representan la coordenada espacial de un elemento de la imagen digital (abreviado píxel) [27].

$$I = f(x, y) \tag{2.1}$$

En DIP, el sensor de imagen convierte una imagen en un número discreto de píxeles y asigna a cada píxel una ubicación numérica (coordenada) y un valor de color o un nivel de gris [27]. La función *f(x,y)* da un solo valor cuando la imagen es en escala de grises (seudo-color) y para una imagen a color da como resultado un vector que representa un color sobre diversos modos o espacios de color, como: RGB, HSL, CMY, y otros [27].

Propiedades básicas de una imagen digital

Una imagen digital tiene tres propiedades básicas, que son:

- **Resolución**

La resolución espacial de una imagen es el número m columnas por el número n filas de píxeles ($m \times n$) [27].

- **Profundidad**
  La profundidad de bits de una imagen es el número de bits utilizados para codificar el valor de un píxel. Para una profundidad de n bits, la imagen tiene una definición (número de tonos o color si nos referinos de forma coloquial) de imagen de 2^n, entonces un píxel puede tener 2^n valores diferentes [27].

- **Número de Planos**
  El número de planos en una imagen corresponde al número de matrices de píxeles que componen la imagen [27].

Iluminación

Es un elemento de gran importancia que suele afectar la complejidad de los algoritmos de visión artificial. Determina la calidad de la imagen adquirida, facilitando el procesamiento posterior de las imágenes digitales. La iluminación de una escena o ambiente se puede realizar con fuentes de luz natural o artificial; aunque la iluminación general del entorno no suele ser aceptable ya que produce imágenes con muy bajo contraste, reflexiones especulares, sombras y detalles falsos. Las técnicas de iluminación más frecuentes son la iluminación difusa, posterior, estructural, direccional y frontal. Las fuentes de iluminación más comunes son las luminarias halógenas, xenón, incandesentes, fluorescentes, láser, flashes, LEDs y fibra óptica.

Captación y adquisición de la imagen digital

Esta etapa tiene como objetivo adquirir una imagen digital, para lo cual se necesita de un dispositivo físico sensible a la luz (transductor que capta la radiación luminosa reflejada en los objetos), produciendo como salida una señal eléctrica. Adicionalmente, se necesita de un instrumento digitalizador, que permita convertir la señal eléctrica del sensor a un formato digital. Esto se logra haciendo uso de cámaras digitales como: cámaras web, cámaras ip, cámaras de sensores CCD (*Charge Coupled Device*), CMOS (*Complementary Metal Oxide Semiconductor*) u otros dispositivos que permitan la captura de imágenes y la digitalicen, como ordenadores y dispositivos móviles. También suele ser necesario, en algunos casos, una tarjeta de adquisición de imágenes, la cual es una interfaz entre el dispositivo capturador y la unidad de procesamiento que generalmente es un computador.

Procesamiento

Los métodos para el procesamiento de imágenes digitales se han desarrollado hacia dos áreas principales de aplicación, como es el mejoramiento de la información pictórica para la interpretación humana y el procesamiento de datos de la imagen para la percepción de una máquina autónoma. El DIP usa como herramienta las matemáticas, donde los algoritmos y las características del ordenador usados tienen un papel muy importante en la manipulación de las imágenes. Existe gran cantidad de métodos o algoritmos utilizados en el DIP, sin embargo, pueden agruparse y representarse en las siguientes etapas.

Pre procesamiento

Esta etapa se encarga de analizar y mejorar la imagen digital obtenida de la adquisición, facilitando la obtención de los resultados en etapas posteriores. Usando algoritmos de DIP se busca generalmente eliminar ruido, reducir el entorno no relevante, mejorar el contraste y extraer regiones de interés. En esta etapa se hace usos de algunas técnicas como escala de grises, filtros, redimensionamiento, ecualización de histogramas, operaciones geométricas, entre otras.

Segmentación

Esta etapa permite subdividir la imagen en las partes que la constituyen, detectando y extrayendo sus objetos. La segmentación de la imagen termina cuando los objetos de interés han sido aislados [28]. Para la segmentación de la imagen se suele usar la técnica de umbral, binarización, erosión, dilatación, apertura, cierre, relleno, filtros, entre otros.

Representación

Esta etapa se encarga de representar o describir los objetos o partes de la imagen obtenidos de la etapa de segmentación, seleccionando y extrayendo características apropiadas para su futura identificación. La representación de objetos se realiza en términos de sus características externas, es decir su contorno, y en términos de sus características internas, que se refiere a los píxeles que comprenden dicho objeto o región. Generalmente se usa la representación externa cuando son de mayor interés las características de forma y una representación interna cuando el interés se centra en las propiedades de reflectividad (color y textura). Las técnicas que suelen emplearse son los códigos cadena, aproximaciones poligonales, firmas, lados de contorno, momentos invariantes de Hu y esqueleto de una

región [28].

Reconocimiento

Esta etapa final permite identificar objetos o formas y definir a que categoría u objeto predefinido (patrones) pertenecen. Los métodos aplicados generalmente son:

- La decisión teórica, se basa en la representación de patrones en forma vectorial y posteriormente en la búsqueda de aproximaciones que permitan asignar estos patrones vectoriales a las diferentes clases de patrones. Las principales técnicas usadas en este método son los clasificadores de mínima distancia, los correladores, los clasificadores de Bayes y las redes neuronales [28].
- Los métodos estructurales, en este caso los patrones se representan en forma simbólica (cadenas, árboles), las técnicas usadas se basan en el emparejamiento de símbolos o en modelos que tratan a los patrones de símbolos como sentencias de un lenguaje artificial [28].

Los algoritmos de las etapas del DIP conforman un gran grupo de métodos matemáticos, cuyo análisis resulta extenso y se encuentra fuera del objetivo principal del presente trabajo de titulación.

Periféricos

Son equipos o dispositivos (actuadores) que reciben información del computador para cumplir con una función específica dentro de un proceso. En el presente trabajo de titulación el dispositivo actuador es el robot DaNI.

2.2.3. Aplicaciones

Gracias al desarrollo tecnológico en los campos de captura y adquisición de imágenes, y los sistemas de cómputo, las aplicaciones de visión artificial se han extendido a áreas como medicina, industria, militar, agricultura, manipulación fotográfica, seguridad, robótica entre muchas otras más. A continuación, se mencionará algunas de las aplicaciones de la visión artificial.

- Reconocimiento y localización de objetos concretos en imágenes digitales
- Clasificación de objetos

- Reconocimiento de rostros
- Reconocimiento de huellas dactilares
- Reconocimiento óptico de caracteres
- Navegación en robótica
- Guiado automático en máquinas
- Determinar trayectorias para un vehículo
- Video vigilancia
- Conteo (personas, animales, objetos y seres microscópicos)
- Control de tráfico (vehícular)
- Control de procesos
- Control de calidad

2.3. Detección y evasión de obstáculos

La detección de obstáculos se realiza mediante la observación y análisis del área en donde se encuentra el robot DaNI. Este proyecto desarrolla los siguientes métodos:

- La combinación de iluminación estructurada y adquisición de imágenes permite detectar la existencia de obstáculos en las rutas del UGV. Su desarrollo se presenta en el Capítulo 3.
- Usando el sensor de ultrasonido se detecta la existencia de objectos a una distancia determinada. Este desarrollo se presentará en el Capítulo 3.

La evasión de los obstáculos se realiza usando visión artificial, mediante el procesamiento de una imagen digital con el modulo LabVIEW NI Vision Development y la implementación de los métodos de planificación de trayectorias, cuya ejecución será requerida luego de haberse detectado la existencia de algún obstáculo. Con esta información el robot DaNI direcciona su ruta de acuerdo a la trayectoria de navegación. El desarrollo correspondiente se presenta en el Capítulo 3 y 4. Este método de evasión de obstáculos también se considera como un método de detección de obstáculos.

Capítulo 3

Desarrollo del UGV mediante NI LabVIEW

3.1. Introducción

LabVIEW es un lenguaje y entorno de programación gráfico (lenguaje G), desarrollado por National Instruments, que permite crear aplicaciones de forma rápida y sencilla [29]. LabVIEW ha sido diseñado para ingenieros y científicos para desarrollar aplicaciones de pruebas, control y medidas. Su naturaleza intuitiva de programación gráfica lo hace fácil de usar por educadores, estudiantes e investigadores para incorporar el software a varios cursos y aplicaciones [30].

3.2. LabVIEW para visión artificial

LabVIEW consta de diversos módulos que facilitan el desarrollo de aplicaciones en diversas áreas; el módulo LabVIEW NI Vision Development es uno de ellos. A continuación se presenta el desarrollo del procesamiento de captura y procesamiento de imágenes.

3.2.1. Captura y Adquisición de la Imagen

En el desarrollo del presente proyecto de grado, para capturar y adquirir una imagen se usa un dispositivo móvil inteligente (*smartphone* [1]) y una aplicación [2] para el mismo.La aplicación permite usar la cámara del dispositivo móvil como

[1] Características principales del *smartphone*: sistema operativo *Android 4.2.2*, velocidad de procesamiento 1700MHz-2 núcleos, RAM 1.5GB-533MHz, cámara de 1920×1080 píxeles, Wi-Fi.

[2] Aplicación **DroidCam** para el *smartphone* y **DroidCam Cliente** para el ordenador

una WEB cam o una cámara IP. Se han probado versiones demo de diversas aplicaciones hasta obtener la de mejores prestaciones, de acuerdo a los requerimientos y limitaciones del dispositivo móvil como permitir usar la cámara mediante *software* LabVIEW y ser compatible con la versión de *android*. La aplicación elegida tiene limitación en la resolución de la imagen capturara, a valores de 320×240 píxeles y tipo RGB. Con una baja resolución se pierde información del entorno (detalles) y en consecuencia se debe limitar el campo de visión de la cámara.

Mediante el VI Vision Acquisition y con la aplicación de cámara IP ya instalada ,en el dispositivo móvil como en el ordenador, el reconocimiento y ajustes de la cámara se torna sencillo. Seleccionamos la cámara que se usará (cam7:DroidCam), seguido se escoge el tipo de adquisición para la imagen y por último la resolución de la misma. Los pasos mencionados se aprecian en las Figuras 3.1, 3.2 y 3.3.

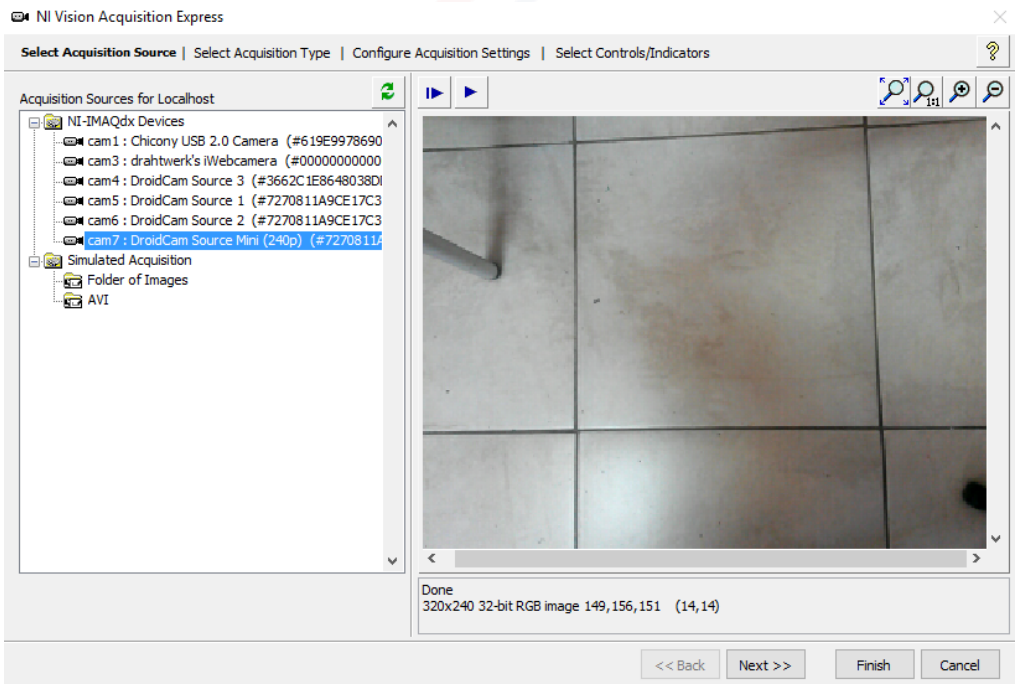


Figura 3.1: Vision Acquisition: Seleccionar la cámara

Realizado la configuración anterior, ya se tiene disponible una imagen digital para ser guardada o usada en diferentes aplicaciones.

3.3. DIP con LabVIEW

Con la imagen obtenida en la sección anterior (Figura 3.4), se procede ha realizar el DIP para tres eventos en particular:

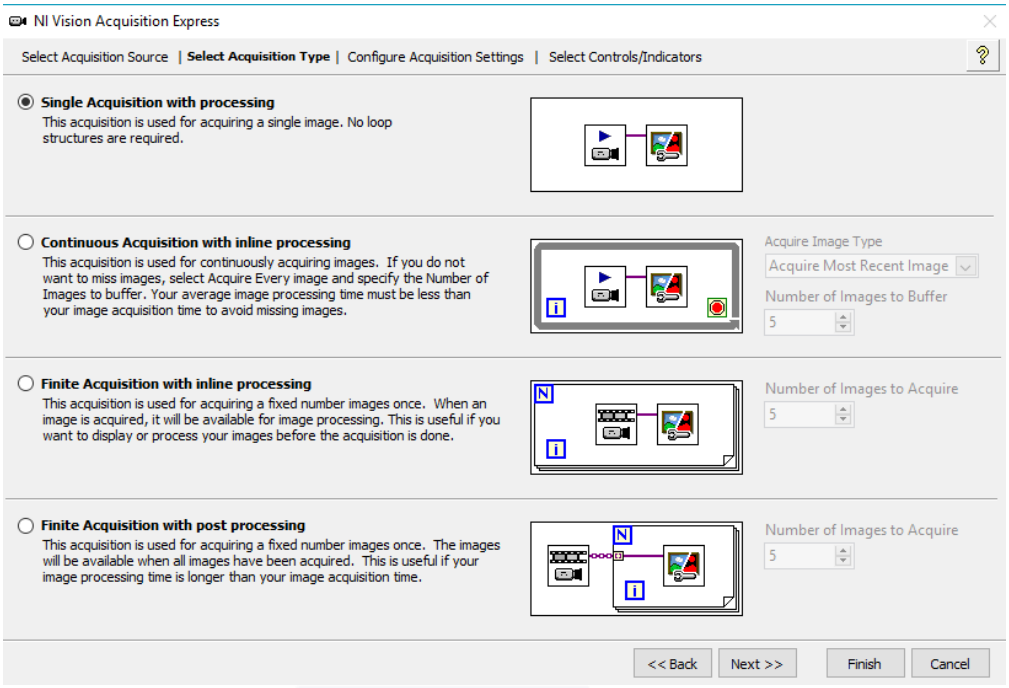


Figura 3.2: Vision Acquisition: Seleccionar el modo de adquisición

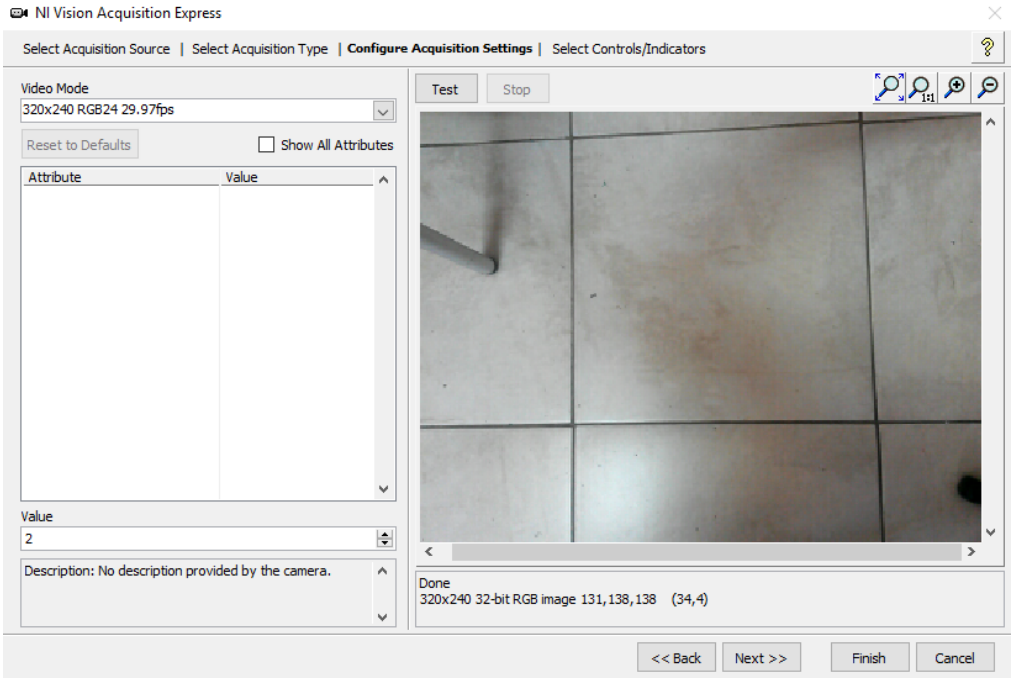


Figura 3.3: Vision Acquisition: Ajustes de adquisición

1. Reconocimiento y localización del robot DaNI
2. Mejoramiento de la imagen
3. Reconocimiento del patrón de iluminación estructurada

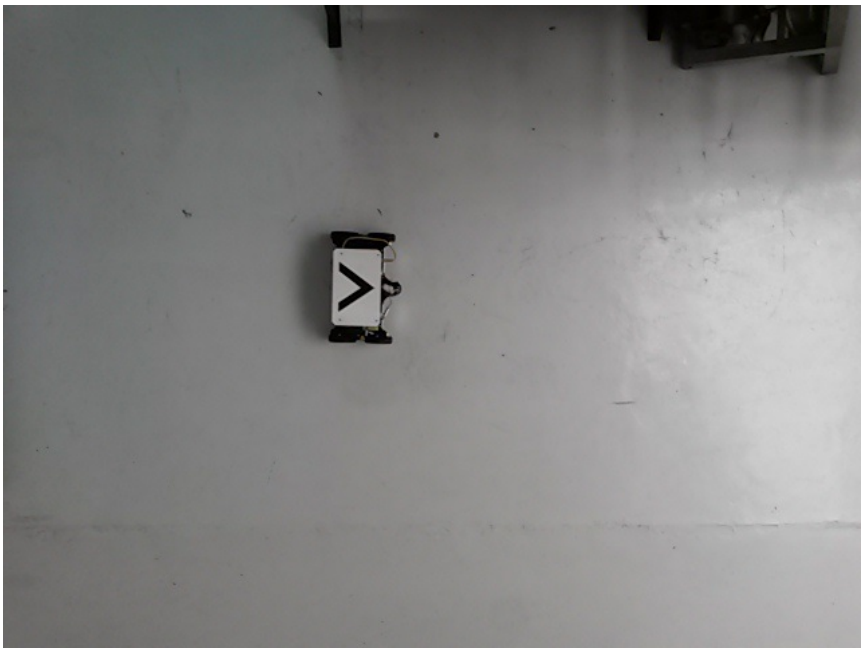


Figura 3.4: Imagen original antes del DIP

3.3.1. Reconocimiento y localización del robot DaNI

El primer paso, desarrollado para este proyecto, consiste en el redimensionamiento de la imagen, de forma que se procese una imagen más pequeña y demande menor costo computacional. Seguidamente, se transforma la imagen a escala de grises de igual modo para reducir el tiempo de procesamiento, pero en especial debido a las técnicas de DIP que solamente son aplicables a estas o a imágenes binarias. Luego se procede a filtrar la imagen para destacar detalles. Se realiza una plantilla o representación de DaNI. Por último, se realiza un pareo(encontrar la similitud o correspondencia de de patrones), que permite localizar a DaNI dentro de la imagen.

Filtrado de la imagen

Disponiendo de la imagen en escala de grises, se procede ha realizar un filtro de mediana. Este es un filtro paso bajo, que asigna a cada píxel la el valor de

la mediana de su vecindad, eliminando de forma efectiva píxeles aislados y reduciendo el detalle, pero no borra el contorno de los objetos como sucede al aplicar un filtro pasa bajo [27]. La Figura 3.5 presenta la imagen en escala de grises luego de ser filtrada.

Figura 3.5: Imagen resultante de aplicar el filtro de mediana

Realce de detalles

Se realiza el realce de detalles aplicando un filtro Laplaciano. Este es un filtro lineal[3] o convolución, el cual resalta la variación de la intensidad luminosa que rodea a un píxel. Este filtro permite extraer el contorno de los objetos y destacar los detalles [27].

El filtro convolución Laplaciano usa el siguiente modelo de *kernel*.

$$\begin{bmatrix} a & d & c \\ b & x & b \\ c & d & a \end{bmatrix} \tag{3.1}$$

Donde a, b, c y d son enteros. Si el coeficiente central es mayor que la suma de los valores absolutos de los coeficientes exteriores, el filtro Laplaciano presenta una imagen en donde se destacan las variaciones de intensidad luminosa; pero uno de sus efectos es que puede generar ruido en la imagen [27].

[3]El filtro lineal reemplaza cada píxel por una suma ponderada de sus vecinos. La matriz que define la vecindad del píxel también especifica el peso asignado a cada vecino. Esta matriz se llama el *convolution kernel* (núcleo de convolución) [27].

La Figura 3.6 presenta el resultado del filtro aplicado a la imagen, donde se aprecia claramente un resalte de las características de la imagen, y en general del símbolo de la parte superior de DaNI.

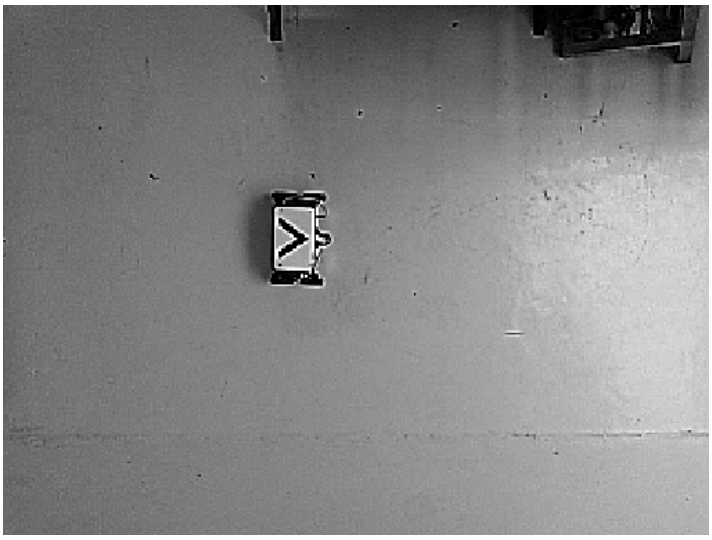


Figura 3.6: Filtro convolución Laplaciano aplicado a DaNI

Ubicación del robot DaNI

Para obtener la ubicación del robot DaNI en la imagen digital, se utilizan algoritmos de correspondencia de patrones (*pattern matching*). Permitiendo localizar regiones de una imagen en escala de grises que coincida con una plantilla predeterminada. La *pattern matching* crea un modelo o plantilla que representa al objeto que está buscando. Entonces, la aplicación de visión artificial busca el modelo en cada imagen adquirida, midiendo la similitud del modelo con algún patrón de píxeles encontrado en la imagen [27].

El método usado es el modelado geométrico de imágenes, éste genera información sobre las características de una imagen de plantilla. Es una técnica de comprensión de la imagen para interpretar la información de la plantilla, y luego usar esta información para encontrar la plantilla en la imagen [27].

El algoritmo de correspondencia geométrica en NI Vision localiza regiones en una imagen a escala de grises que coincida con una plantilla o modelo de un patrón de referencia. La correspondencia geométrica está especializada para localizar plantillas que se caracterizan por geométricas distintas o información de

forma. La correspondencia geométrica encuentra coincidencias de la plantilla independientemente de la variación de iluminación, desenfoque, ruido, oclusión y transformaciones geométricas tales como desplazamiento, rotación, o escala de la plantilla. La correspondencia geométrica devuelve la ubicación del centro de la plantilla, su orientación y la escala [31]. Para facilitar reconocimiento y ubicación de DaNI, se ha realizado un símbolo en su parte superior (Figura 3.7).

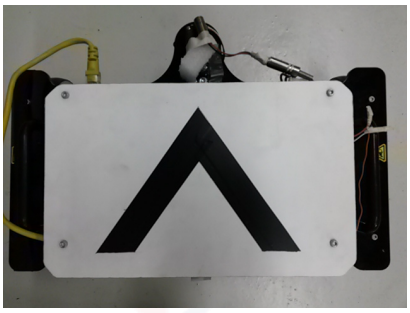


Figura 3.7: Símbolo en la parte superior de DaNI

Usando la función *Geometric Matching*, de la pestaña funciones de visión máquina, se procede a hacer la descripción de objeto que será modelo para el reconocimiento de patrón, en este caso el robot DaNI. Esta misma función posteriormente permitirá el reconocimiento de DaNI.

Primeramente se obtiene la plantilla modelo o representación de DaNI. Dando clic en opción *New Template* crearemos una plantilla de la imagen ,o una región de ella, que se encuentre siendo procesada. La Figura 3.8 muestra la selección de DaNI como plantilla.

A continuación, se especifican los parámetros de curva para la plantilla creada (Figura 3.9), en los cuales se ha seleccionado para el borde un modo de extracción normal, un umbral de 175 y filtro fino. Finalmente se guarda esta plantilla, la cual puede ser modificada posteriormente según las necesidad y usada para hacer el reconocimiento del robot DaNI.

El reconocimiento de DaNI se realiza mediante la misma función. Al ejecutar el algoritmo ahora se obtendrá datos como número de objetos localizados, posición y rotación de los mismos respecto a la descripción de la plantilla de DaNI realizada anteriormente. En las Figuras 3.10 y 3.11 se presentan unas pruebas

Figura 3.8: Selección de plantilla para DaNI

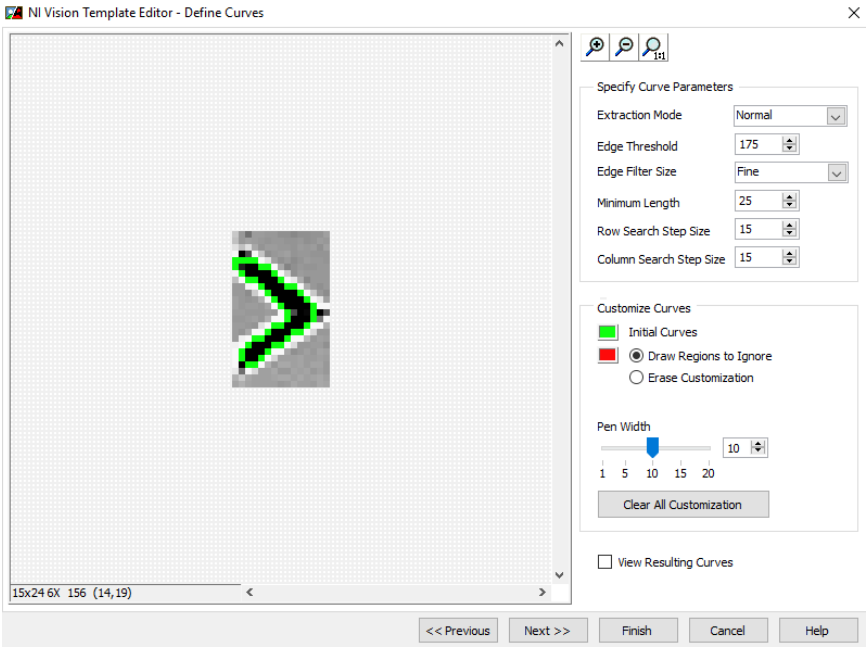


Figura 3.9: Parámetros de curva para la plantilla de DaNI

realizadas y sus respectivos resultados (Tabla 3.1).

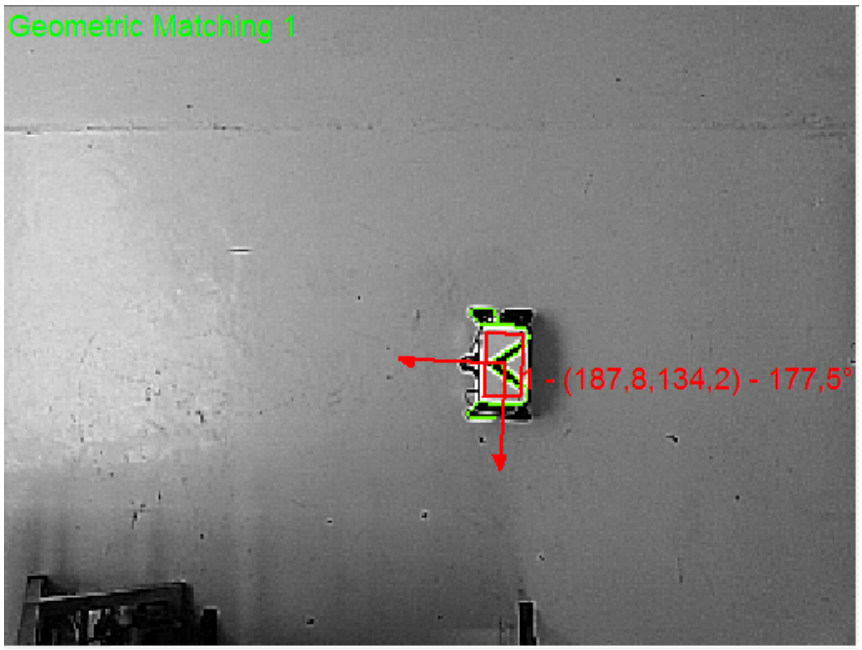


Figura 3.10: Reconocimiento de DaNI (prueba 1)

Tabla 3.1: Resultados de pruebas de reconocimiento de robot DaNI

Prueba No.	**Posición x [píxeles]**	**Posición y [píxeles]**	**Ángulo [grados]**
1	187.76	134.16	177.56
2	208.23	133.17	93.30

3.3.2. Mejoramiento de la imagen

A continuación se presenta la metodología utilizada para el mejoramiento de la imagen adquirida, y su segmentación, de forma que se obtenga una imagen binaria. La imagen obtenida posteriormente es procesada en MATLAB mediante algoritmos de generación de ruta óptima (Capítulo 4), obteniendo así la trayecto que debe realizar DaNI.

En la imagen pre-procesada, se realiza extracción de bordes, detalles y objetos presentes en la imagen. La Figura 3.12a y 3.12b presentan la imagen en escala de grises y la imagen filtrada respectivamente.

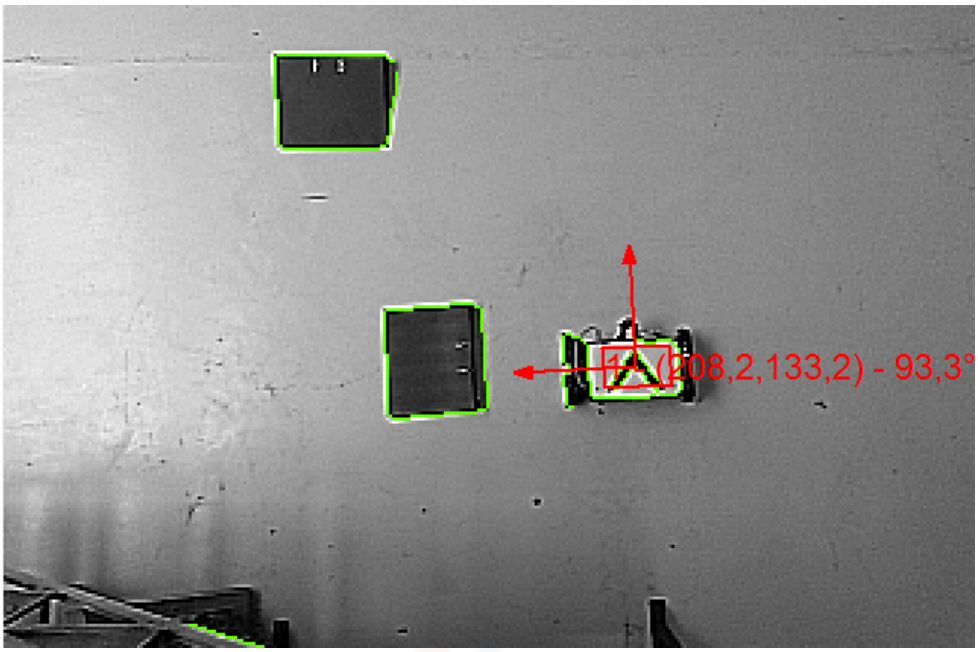


Figura 3.11: Reconocimiento de DaNI (prueba 2)

(a) (b)

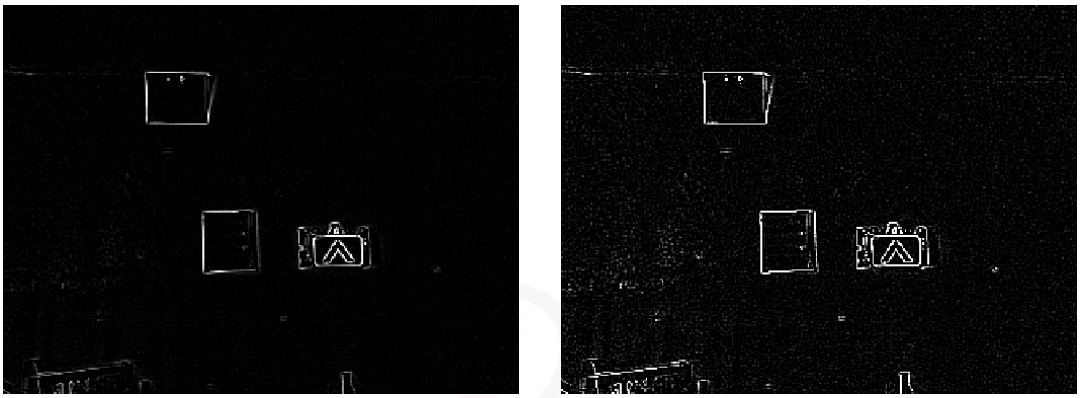
Figura 3.12: Aplicación del filtro de la mediana

Extracción de bordes

La extracción y resaltamiento de bordes se realiza a través de un filtro Laplaciano. Para extraer los bordes de la imagen el coeficiente central del kernel del filtro debe ser igual a la suma de los valores absolutos de los coeficientes exteriores. El filtro Laplaciano extrae los píxeles donde se encuentran variaciones significativas de la intensidad de luz debido a la presencia de bordes afilados, límites entre objetos, modificación en la textura de un fondo, ruido u otros efectos.

La nueva imagen presentará contornos blancos sobre un fondo negro [27].

La Figura 3.13a y 3.13b presentan el resultado de la extracción de bordes y realce de bordes mediante el filtro convolución Laplaciano.



(a) (b)

Figura 3.13: Extracción y realce de bordes mediante el filtro convolución Laplaciano

Umbralización de la imagen

El umbral consiste en segmentar una imagen en dos regiones: una región partículas y una región de fondo. Permite aislar objetos de interés en una imagen. También convierte una imagen de escala de grises a una imagen binaria. El proceso de umbral funciona ajustando a 1 todos los píxeles que pertenecen a un intervalo de nivel de gris, denominado intervalo de umbral, y establece todos los demás píxeles de la imagen a 0. El intervalo de umbral se define por dos parámetros (umbral inferior y umbral superior). Todos los píxeles que tienen valores de nivel de gris iguales o mayores que el umbral inferior e iguales o menores que el umbral superior se seleccionan como píxeles pertenecientes a las partículas de la imagen. Todos los demás píxeles se consideran parte del fondo [27].

Un problema crítico y frecuente en segmentar una imagen en partículas y regiones de fondo se produce cuando los límites no se encuentran claramente demarcados. En tal caso, la elección del umbral correcto se convierte en algo subjetivo. Por lo tanto, es recomendable mejorar las imágenes antes de umbralizar para delinear en dónde se encuentran los bordes correctos.

La Figura 3.14 presenta el resultado de la imagen umbralizada para un intervalo de valores de píxeles mayores o iguales a 127. Este valor escogido corresponde a un rango de valores desde medios a claros y presentó mejor resultado al realizarse algunas pruebas. El intervalo de umbral varía dependiendo del entorno (principalmente contraste, piso, objetos e iluminación), y de su correcta elección depende también la calidad e información de la imagen binaria obtenida.

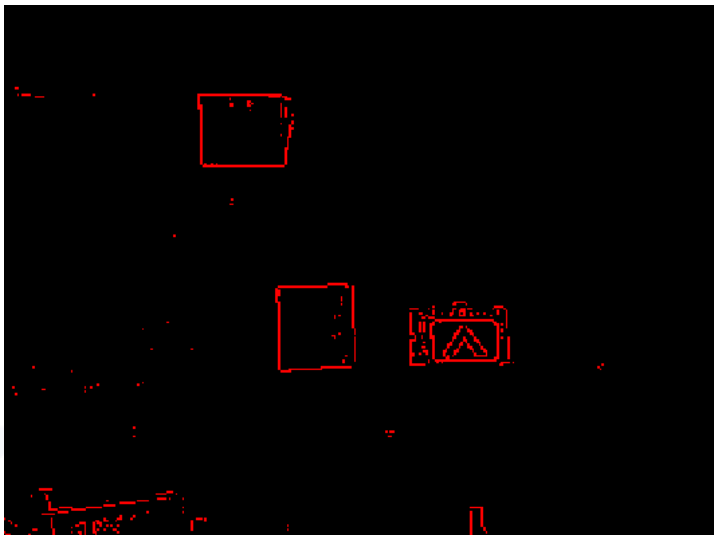


Figura 3.14: Resultado obtenido de umbralizar la imagen

Transformaciones morfológicas

Por último, se realizan algunas transformaciones morfológicas que permiten extraer y alterar la estructura de las partículas binarias y, por tanto, corregir información no deseada (como partículas de ruido, partículas que tocan el borde de las imágenes, partículas que se tocan entre sí y partículas con bordes irregulares) que ocurriese durante la umbralización.

Los operadores morfológicos que cambian la forma de las partículas procesan un píxel basado en su número de vecinos y los valores de esos vecinos. Se usa una máscara binaria 2D llamada elemento estructurante para definir el tamaño y el efecto del vecindario en cada píxel, controlando el efecto de las funciones morfológicas binarias sobre la forma y el borde de una partícula. El tamaño y el contenido del elemento estructurador especifica los píxeles que una operación morfológica toma en cuenta al determinar el nuevo valor del píxel que se está

procesando. El elemento estructurante debe tener un eje de tamaño impar para acomodar un píxel central, que es el píxel que se está procesando. Los factores que influyen en el elemento estructurador y definen qué píxeles procesar son: el tamaño del elemento estructurador, los valores de los sectores de elementos estructurantes y la forma del marco de píxeles [27]. A continuación, se presenta el elemento estructurante más común que es una matriz 3×3 que contiene valores de 1.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.2}$$

Las operaciones morfológicas se las puede clasificar en **primarias** y **avanzadas**.

Las **operaciones morfológicas primarias** trabajan en imágenes binarias para procesar cada píxel basado en su vecindario. Estas operaciones se usan para expandir o reducir partículas, suavizar los bordes de los objetos, encontrar los límites externos e internos de las partículas, cerrar (llenar pequeños agujeros y suavizar los límites) y abrir (eliminar partículas pequeñas y suavizar los límites) partículas, y localizar configuraciones particulares de píxeles. Las funciones morfológicas primarias incluyen tres funciones binarias fundamentales de procesamiento como la erosión, dilatación y *hit-miss*. Otras transformaciones como *opening*, *closing*, *inner gradient*, *outer gradient*, *thinning* y *auto-median* son combinaciones de estas tres funciones [27].

La erosión elimina los píxeles aislados en el fondo y reduce el contorno de las partículas. Donde,

- Si AND $(P_i) = 1$, entonces $P_0 = 1$, sino $P_0 = 0$ [27]

para un píxel P_0 dado, el elemento estructurante está centrado en P_0. Los píxeles enmascarados por un coeficiente del elemento estructurador igual a 1 se denominan P_i.

Por otra parte, la dilatación elimina pequeños agujeros aislados en partículas y expande el contorno de las partículas. Esta función tiene el efecto inverso de una erosión, porque la dilatación es equivalente a erosionar el fondo. Donde,

- Si OR $P_i = 1$, entonces $P_0 = 1$, sino $P_0 = 0$ [27]

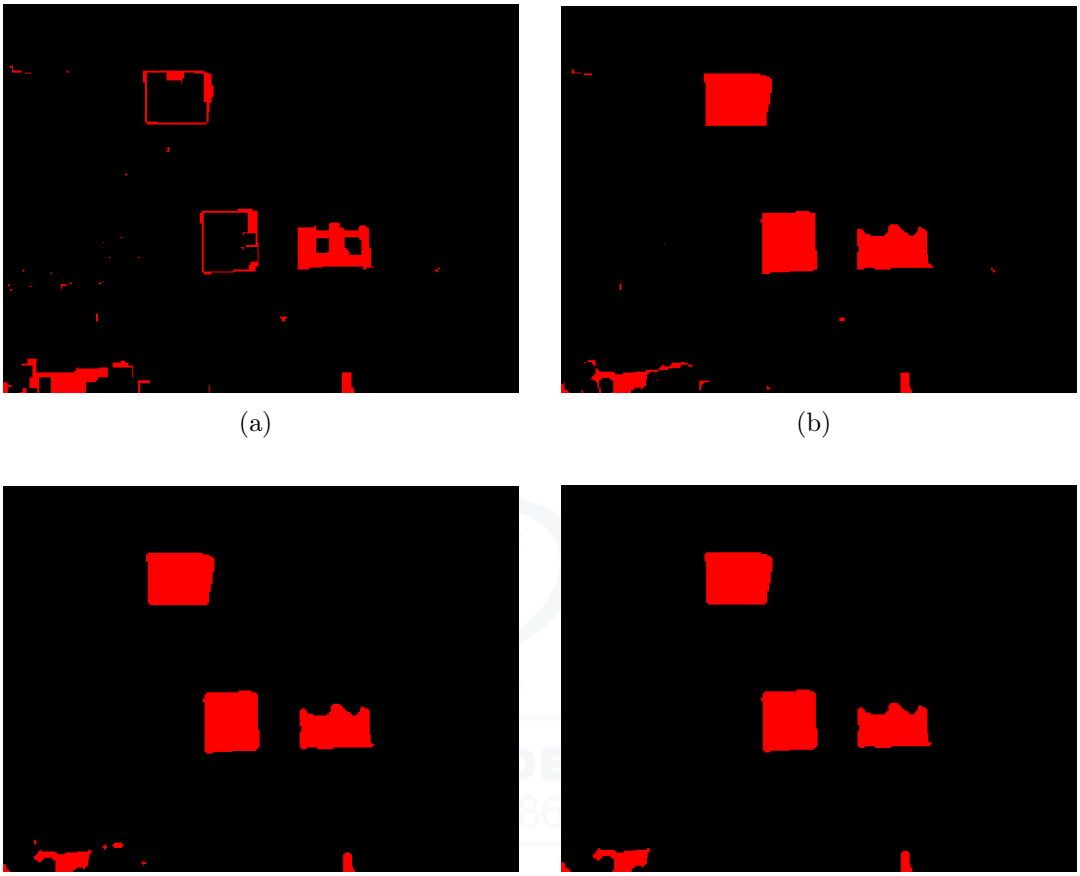
Por último, la función *hit-miss* permite localizar configuraciones particulares de píxeles. Esta función extrae cada píxel situado en un vecindario que coincida exactamente con la plantilla definida por el elemento estructurador. Dependiendo de la configuración del elemento estructurador, la función hit-miss puede localizar píxeles individuales aislados, patrones transversales o longitudinales, ángulos rectos a lo largo de los bordes de partículas y otras formas especificadas por el usuario. Cuanto mayor es el tamaño del elemento estructurador, más específica puede ser la plantilla investigada. En un elemento estructurante con un coeficiente central igual a 0, una función *hit-miss* cambia todos los píxeles establecidos en 1 en la imagen fuente al valor 0 [27].

• Si los píxeles P_i definen exactamente la misma plantilla que el elemento estructurador, entonces $P_0 = 1$, si no $P_0 = 0$. [27]

Las **operaciones de morfología avanzada** se basan en los operadores morfológicos primarios y trabajan sobre las partículas de la imagen. Son combinaciones condicionales de transformaciones fundamentales, como la erosión binaria y la dilatación. Se utilizan para llenar orificios en partículas, eliminar las partículas que tocan el borde de la imagen, eliminar partículas pequeñas y grandes no deseadas, separar partículas en contacto, encontrar el casco convexo de las partículas,entre otras [27].

Tanto, las operaciones morfológicas primarias como las avanzadas se utilizan para preparar las partículas para el análisis cuantitativo, observar la geometría de las regiones, extraer las formas más simples para fines de modelado e identificación.

Para mejorar la imagen del entorno de DaNI, en el DIP se han usado las siguientes funciones morfológicas binarias de la pestaña funciones de procesamiento binario: erosión, dilatación, *closing*, *hole filling* y filtro pasa bajos, para eliminar partículas pequeñas. Los resultados de las mismas se muestran en la Figura 3.15, donde las Figuras 3.15a, 3.15b, 3.15c y 3.15d presentan el entorno de DaNI luego de aplicar las funciones morfológicas binarias de *closing*, *hole filling*, *erode-dilate* y *remove small objects*, respectivamente.



(a) (b)

(c) (d)

Figura 3.15: Operaciones morfológicas binarias

3.3.3. Reconocimiento del patrón de iluminación estructurada

En esta sección se presenta el reconocimiento del patrón dado por un láser. Su procedimiento es similar al realizado en la subsección 3.3.1, pero con una plantilla diferente.

La iluminación estructurada, mediante el uso de láseres u otras fuentes de luz, proyecta patrones conocidos (puntos, franjas o rejillas) sobre la superficie del objeto de interés. La deformación del patrón sobre el objeto de interés tendrá una forma conocida. Comparando la deformación conocida con el detectado en cada caso particular se puede detectar la presencia o ausencia del objeto en la escena. Además, analizando la forma del patrón deformado producido se puede obtener información sobre las características dimensionales del objeto de interés [25]. La

Figura 3.16 presenta una esquematización para detección de un objeto.

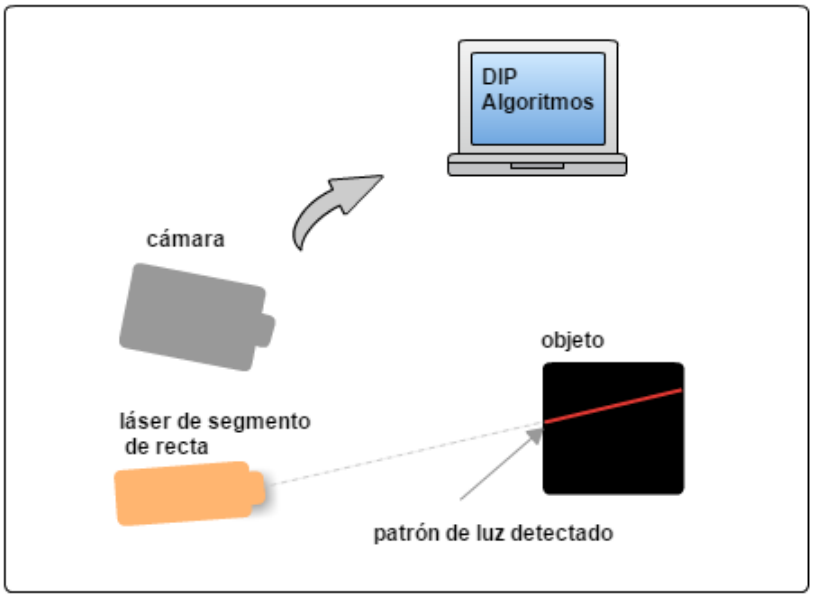


Figura 3.16: Detección de objetos mediante iluminación estructurada

La Figura 3.17 muestra el patrón dado por un láser de luz roja de franja (segmento de recta). Su patrón se presenta como un segmento de recta; en su reconocimiento, para el caso usado, solamente importa si el patrón existe o no en la imagen sin importar su tamaño, posición o dirección. Si el patrón es reconocido (existe en la imagen) significa que existe un objeto frente a DaNI entonces se deben tomar mediadas al respecto en el desenvolvimiento de DaNI, como pausar su desplazamiento o generar una nueva ruta (subsubsección 3.4.2). Las Figuras 3.18 y 3.19 presentan los resultados de pruebas realizadas para dos situaciones distintas. La primera, cuando un objeto se encuentra frente a frente con DaNI. La segunda, cuando el objeto y DaNI se encuentra con direcciones distintas.

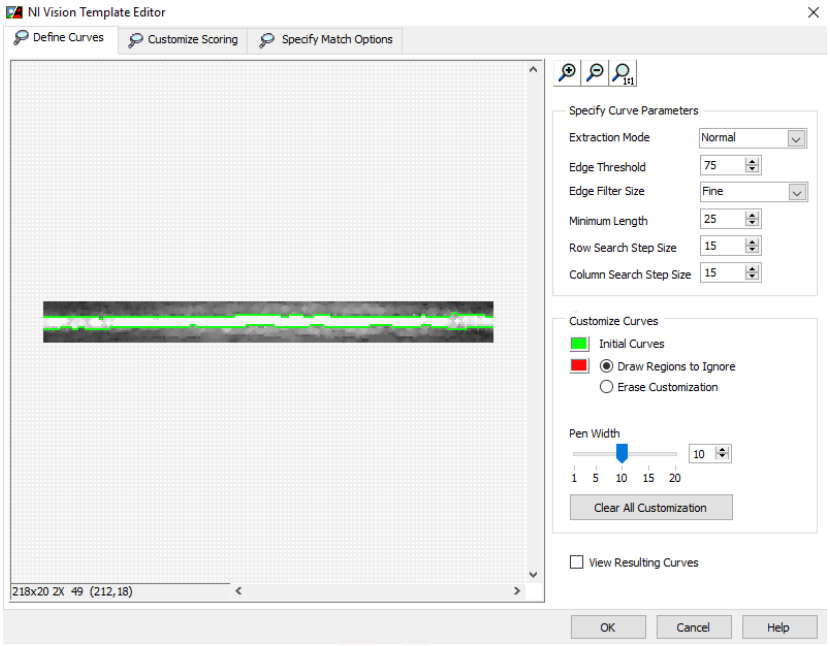
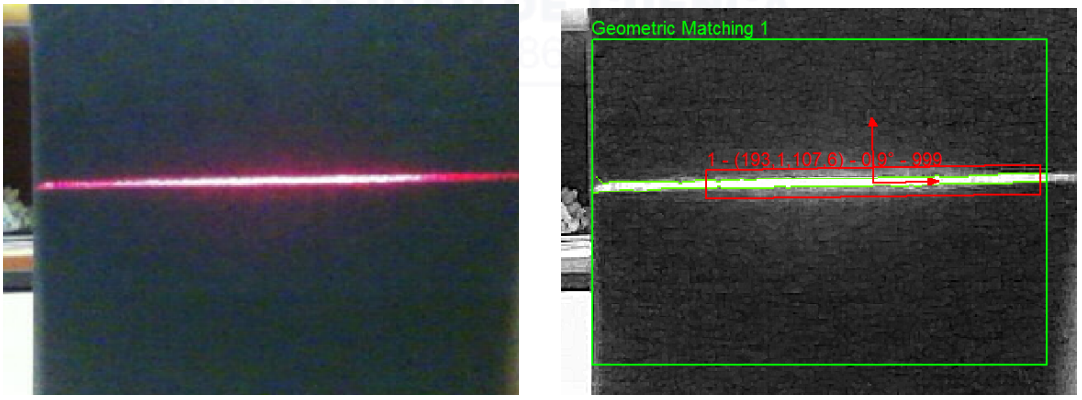
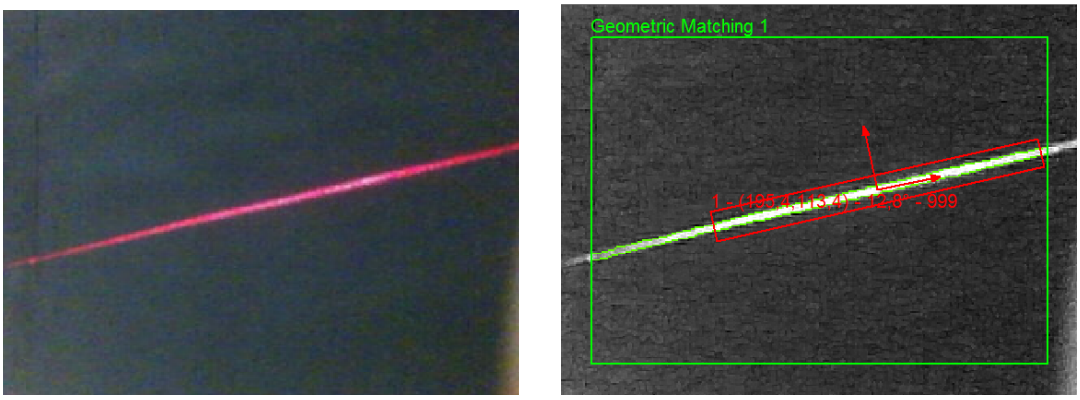


Figura 3.17: Parámetros de curva para la plantilla de la iluminación estructurada



(a) (b)

Figura 3.18: Reconocimiento del patrón dado por iluminación estructurada (situación 1)



(a) (b)

Figura 3.19: Reconocimiento del patrón dado por iluminación estructurada (situación 2)

3.4. Implementación del UGV con DaNI

En esta sección se presenta el desarrollo del UGV con el robot DaNI. Se utilizan los siguientes *toolkits* de LabVIEW: NI LabVIEW 2012 y los módulos LabVIEW Robotics, LabVIEW Real-Time y LabVIEW FPGA. La configuración inicial de DaNI, las pruebas del funcionamiento del sensor de ultrasonido y motores, se realizan fácilmente siguiendo las indicaciones dadas en [24, 32, 33]. Estos tutoriales también presentan algunas prácticas realizadas en la plataforma DaNI mediante LabVIEW Robotics y LabVIEW FPGA.

La comunicación entre DaNI y el ordenador se ha realizado de forma inalámbrica, mediante wifi. Se instaló un *router* inalámbrico sobre DaNI y se configuró al mismo. Se recomienda asignar una dirección de IP estática a DaNI, para evitar inconveniente en comunicación por cambios de dirección IP dinámica.

3.4.1. Desarrollo sobre la FPGA de la NI sbRIO-9631

En la tarjeta NI sbRIO-9631 de DaNI se procede ha realizar la programación necesaria sobre la FPGA que permita controlar y observar la actividad de los motores, *encoders* y sensor ultrasónico de DaNI. La programación respectiva de cada uno de los elementos mencionados se encuentra incluida en las librerías de LabVIEW Robotics, las cuales pueden ser utilizadas y modificadas por los usuarios.

Programación del sensor de ultrasonido

El sensor de ultrasonido implementado en DaNI es transductor *Parallax PING*. Mediante el control de un microcontrolador host *(trigger pulse)*, el sensor emite una ráfaga corta de 40 kHz (ultrasónica). Esta ráfaga viaja por el aire a unos 344,4 m/s, golpea un objeto y luego regresa al sensor. El sensor PING proporciona un impulso de salida al host que termina cuando se detecta el eco; por lo tanto, la anchura de este pulso corresponde a la distancia al objeto localizado [6].

El transductor mide el tiempo transcurrido entre la transmisión y la recepción de una señal válida. *Time-offlight* es el tiempo transcurrido entre la emisión de un paquete de ondas ultrasónicas y la recepción de la reflexión. El tiempo se mide en la FPGA en señales del reloj de 40 MHz de la FPGA, lo que significa que cada 1 tick es igual a 25 nanosegundos. El código FPGA convierte ticks a datos de tiempo de vuelo y luego a distancia en metros [24, 34]. Las características y protocolo de comunicación del sensor se encuentran en [6, 34].

La Figura 3.20 muestra la programación del sensor ultrasónico en la FPGA. El FPGA recibe una señal digital en Port0/DIO6 y reporta la distancia (en metros) mediante el indicador *sensor distance*.

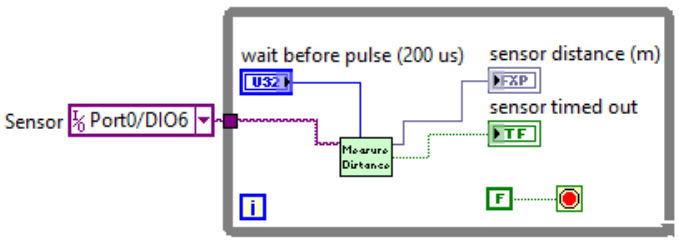


Figura 3.20: Programación del sensor ultrasónico en la FPGA

Control de los motores DC

La programación de los motores en la FPGA se realiza para mantener estable la velocidad de los mismos. La velocidad se especifica en los controles *left ccw velocity setpoint* y *right ccw velocity setpoint*, dadas en (rad/s). Para realizar el control, se utilizan los valores almacenados en los indicadores *left velocity* y *right velocity*, dadas en $pulsos/intervalo$. También se utiliza el valor almacenado en el control *velocity interval*, dado en *ticks*, que se refiere a un encoder. Las velocidades

dadas en *pulsos/intervalo* se transforman a *rad/s* mediante el sub VI *Convert Counts per Interval to Radians per Second*. Los valores obtenidos del sub VI anterior son normalizados en los VIs *Normalize Motor Velocity*; posteriormente son tratados mediante las funciones de control PID (proporcional, integral, derivativo). El algoritmo PID lee la señal del encoder (previamente tratada) y calcula el error entre el valor de velocidad establecida y valores medido. Luego calcula la salida deseada del motor para corregir el error. La señal de salida hacia los controladores de los motores se realiza mediante modulación por ancho de pulsos (PWM, por sus siglas en inglés) hacia los motores izquierdo y derecho por medio de los pines del sbRIO, Port0/DIO5 y Port0/DIO4 respectivamente. La Figura 3.21 muestra la programación en la FPGA para controlar los motores DC de DaNI.

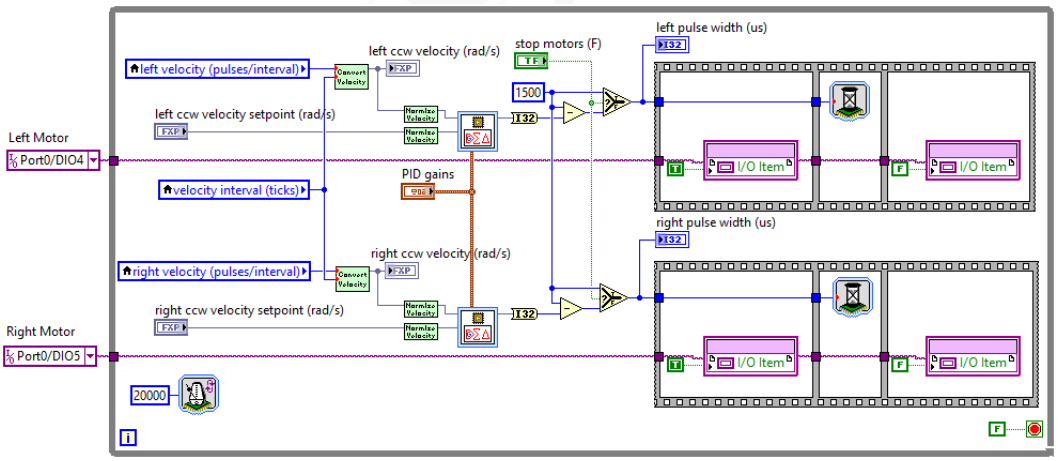


Figura 3.21: Programación en la FPGA para controlar los motores DC de DaNI

Programación de los encoders

Un encoder o codificador es un dispositivo electro mecánico que convierte el desplazamiento lineal o rotativo en señales digitales o de impulsos. Un encoder óptico utiliza un disco giratorio, una fuente de luz y un fotodetector. El disco se encuentra montado en el eje del motor y tiene patrones de sectores opacos y transparentes. La fuente de luz del codificador, apuntando a un fotodetector, pasa a través de los sectores del disco. A medida que el disco gira, estos patrones interrumpen la luz emitida sobre el fotodetector, generando como salida una señal digital o de impulso [24].

Existen dos tipos generales de codificadores: codificadores absolutos e incrementales [24].

El **codificador absoluto** genera un patrón de palabra único para cada posición del eje. Las pistas del disco, generalmente de cuatro a seis, comúnmente están codificadas para generar salidas de código binario.

El **codificador incremental** genera un impulso, en oposición a una palabra digital completa, para cada paso incremental. Aunque el codificador incremental no emite la posición absoluta, proporciona mayor resolución a menor costo. Por ejemplo, un codificador incremental con una única pista de código, denominado codificador de tacómetro, genera una señal de impulso cuya frecuencia indica la velocidad de desplazamiento. Sin embargo, la salida del codificador de un solo canal no indica la dirección. Para determinar la dirección,se usa un **codificador de dos canales o cuadratura** que utiliza dos detectores y dos pistas de código.

El codificador incremental (más común) utiliza dos canales de salida (A y B) y dos pistas de código con sectores desfasados 90^o (Figura 3.22). Los dos canales de salida indican tanto la posición como el sentido de giro. Por ejemplo, si ocurre primero un pulso en A y luego en B, el disco está girando en sentido horario. Si tiene lugar primero un pulso en B y luego en A, entonces el disco está rotando en el sentido inverso a las agujas del reloj. Por lo tanto, si se monitoriza tanto el número de pulsos como la fase relativa de las señales A y B, se puede hacer un seguimiento de la posición y de la dirección de la rotación. Algunos detectores de cuadratura incluyen un tercer canal de salida, denominado señal cero o de referencia, que suministra un único impulso por revolución que puede utilizarse para la determinación precisa de una posición de referencia [24].

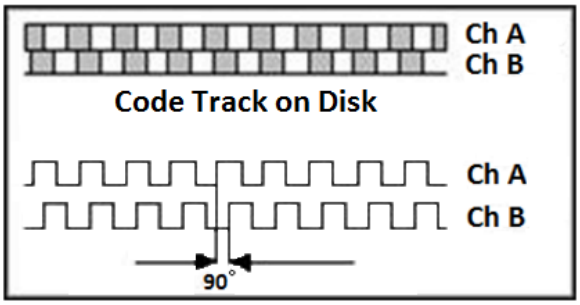


Figura 3.22: Encoder de cuadratura de dos canales (A y B)

Para determinar la velocidad angular y la aceleración de las ruedas mediante

las señales proporcionadas por el encoder de cuadratura, se cuenta el número de pulsos en un intervalo de tiempo dado (Figura 3.23).

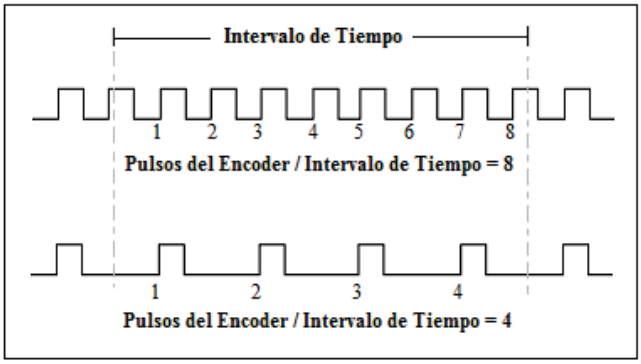


Figura 3.23: Estimación de velocidad mediante señales del encoder de cuadratura

Para calcular la velocidad angular se utiliza la siguiente Ecuación 3.3

$$velocidad = \frac{\frac{pulsos\,del\,encoder}{pulsos\,por\,revolución} \times \frac{60\,sec}{1\,min}}{intervalo\,de\,tiempo\,fijo\,[seg]}[rpm] \tag{3.3}$$

Donde, los pulsos del encoder es el número de pulsos dados por el encoder de cuadratura en un intervalo de tiempo fijo.

Para estimar la aceleración se puede usar la siguiente Ecuación 3.4 :

$$aceleración = \frac{\frac{pulsos\,del\,encoder_n - pulsos\,del\,encoder_{n-1}}{pulsos\,por\,revolución}}{(intervalo\,de\,tiempo\,fijo)^2\,[sec^2]} \times \left(\frac{60\,sec}{1\,min}\right)^2 [rpm^2] \tag{3.4}$$

El codificador montado sobre DaNI es un encoder de cuadratura, que produce 100 conteos/revolución (CPR) y 400 pulsos/revolución (PPR). Estos permiten realizar una lógica de control para que el robot se mueva a una distancia fija, gire a una posición específica, se mueva a una velocidad constante, etc.

El desplazamiento por conteos y pulsos se obtiene realizando las siguientes operaciones (Ecuaciónes 3.5 y 3.6).

$$\frac{0{,}3192\,m/rev}{100\,CPR} = 0{,}003192\,[m/ciclo] \tag{3.5}$$

$$\frac{0{,}3192\,m/revolución}{400\,PPR} = 0{,}000798\,[m/pulso] \tag{3.6}$$

Donde, el dato 0.3192 m corresponde al perímetro (circunferencia) de las ruedas de DaNI.

Considerando los 400 pulsos/revolución y la longitud de la circunferencia de la rueda de DaNI, la resolución para recorrer una distancia en forma lineal es de 0,000798 m o 0,798 mm.

En la Figura 3.24 se presenta la programación de enconders de cuadratura sobre la FPGA. Se realiza lectura de los pulsos generados correspondientes a cada motor a través del subVI *Increment Encoder Count*, el cual incrementa o decrementa un contador dependiendo si el giro es horario o antihorario. Los pines digitales de E/S Port0/DIO0 y Port0/DIO1 corresponden al encoder de la rueda izquierda, fase A y fase B respectivamente. El Port0/DIO2 y Port0/DIO3 corresponde al encoder derecho, fase A y fase B respectivamente. En cualquier instante, los pulsos y velocidades pueden ser leidas en los indicadores *Pulses (Left Wheel)* y *Pulses (right Wheel)*, y *left velocity* y *right velocity*, respectivamente.

3.4.2. Desarrollo del UGV con DaNI mediante el módulo LabVIEW Robotics

El desarrollo (su programación) del UGV con DaNI se ha realizado mediante el módulo LabView Robotics. Se emplean los algoritmos presentados en las Secciones 3.2, 3.3 y la Subseccion 3.4.1 que participan en la autonomía de DaNI. El programa desarrollado implementa las siguientes acciones:

1. Capturar y adquirir la imagen del entorno en que DaNI se desplaza
2. Obtener la posición inicial de DaNI dentro del entorno
3. Procesar (mejorar) la imagen del entorno
4. Definir la posición final deseada de DaNI dentro de su entorno
5. Generar la ruta óptima para el desplazamiento de DaNI
6. Desplazamiento de DaNI sobre la ruta óptima
7. Detección y evasión de obstáculos (que no se encontraban presentes al generar la ruta óptima)

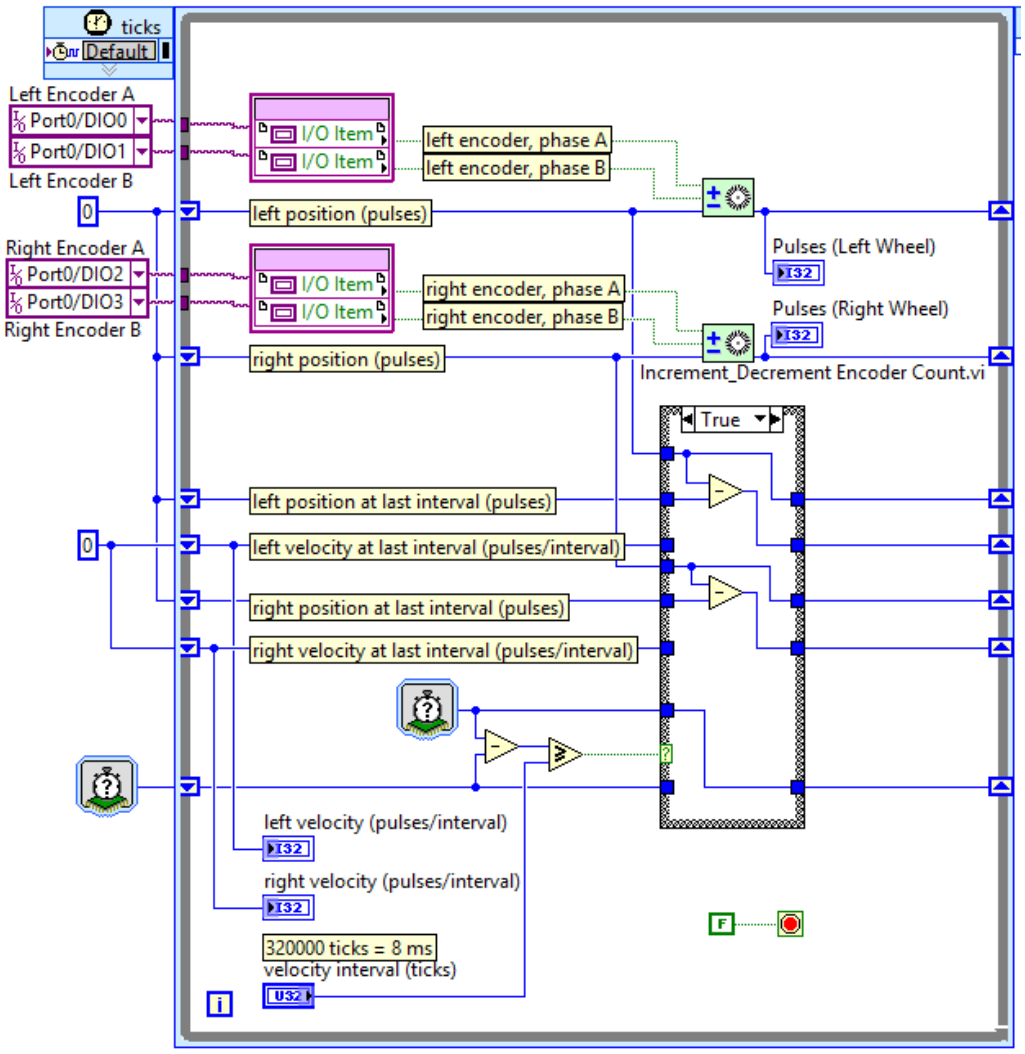


Figura 3.24: Programación de enconders de cuadratura sobre la FPGA

El programa desarrollado para dotar de autonomía a DaNI se encuentra representado mediante el diagrama flujo de las Figuras 3.25 y 3.26. A continuación se realiza una breve explicación de las partes que conforman el programa del UGV DaNI.

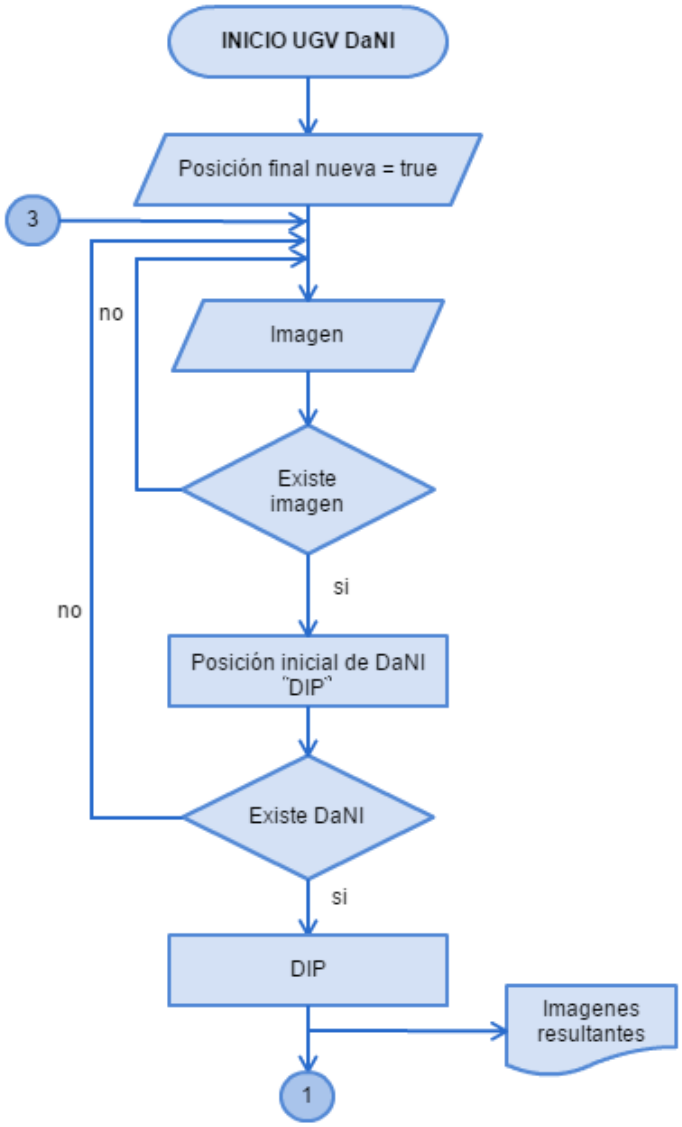


Figura 3.25: Diagrama de flujo del UGV DaNI

Capturar y adquirir la imagen del entorno en que DaNI se desplaza

Su función es capturar y adquirir una imagen digital para un posterior procesamiento de la misma (Subsección 3.2.1). A este algoritmo se encuentra dentro de una estructura de secuencia, siendo el primero en ejecutarse.

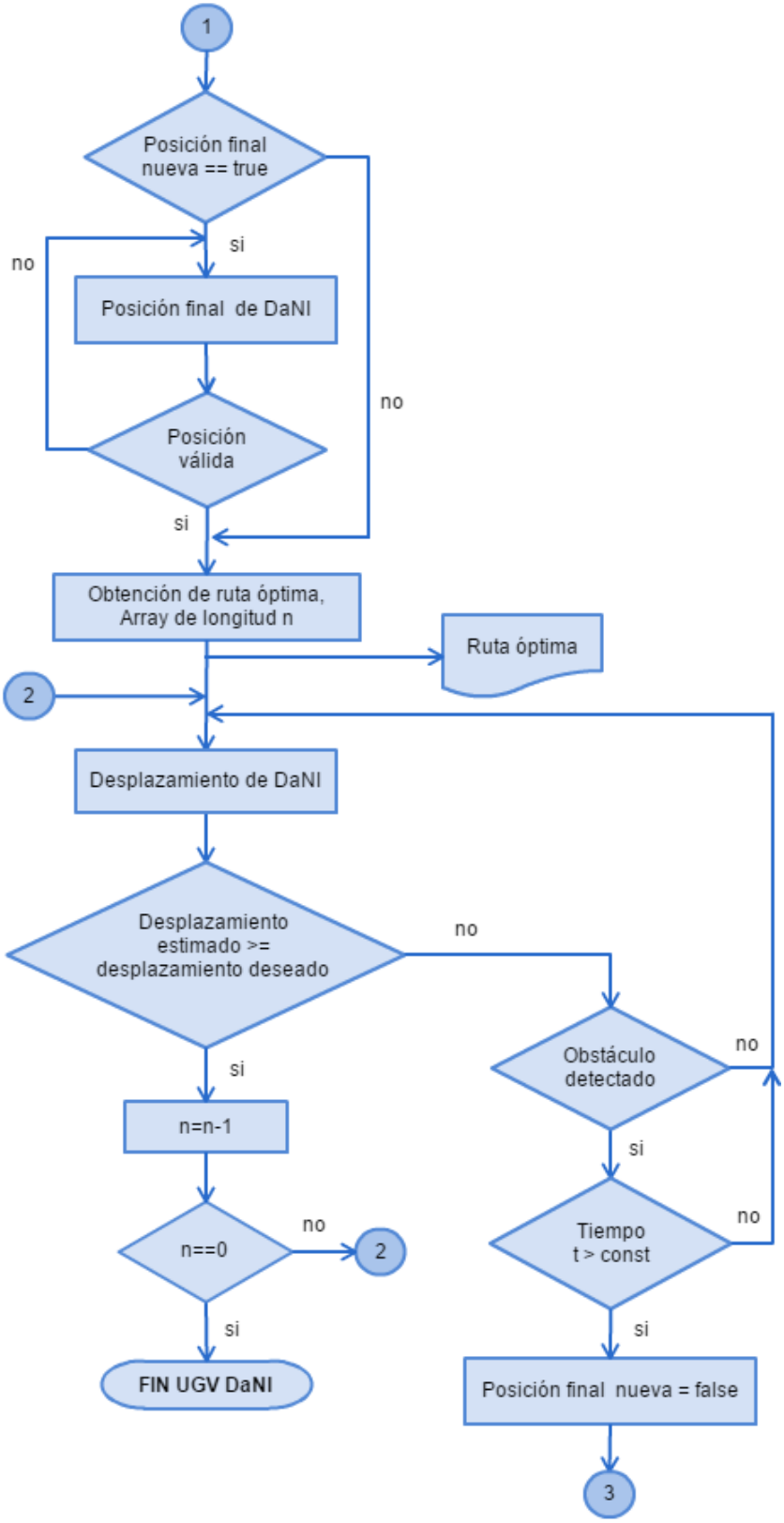


Figura 3.26: Diagrama de flujo del UGV DaNI

Obtener la posición inicial de DaNI dentro del entorno

El desarrollo del algoritmo se presenta en la Subsección 3.3.1. En caso de no obtener alguna poción, se realiza nuevamente la captura y adquisición de la imagen. La coordenada de la ubicación de DaNI (posición inicial) se usa posteriormente en el calculó de ruta optima (Capítulo 4).

Mejorar la imagen del entorno

El desarrollo de la misma de presenta en la Subsección 3.3.2. Genera una imagen binaria que contiene solamente información relevante del entorno de DaNI. La imagen binaria es el entorno en que se realiza la obtención de la ruta óptima (Capítulo 4). Los tres algoritmos han sido implementados en un subVI. Se encuentran dentro de una estructura de secuencia y dentro de un bucle *while* que mantiene en ejecución la estructura secuencial hasta obtener la ubicación de DaNI (Figuras 3.27 y 3.28).

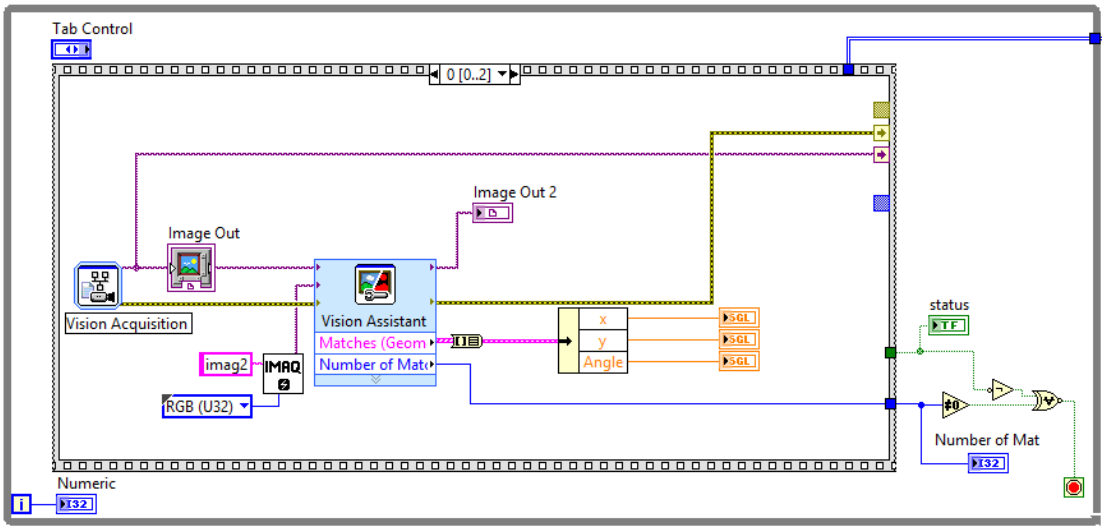


Figura 3.27: Diagrama de Bloques de Adquisición y Ubicación de DaNI

Definir la posición final deseada de DaNI dentro de su entorno

La posición final deseada de DaNI se obtiene mediante los siguiente bloques de LabVIEW que se presentan en la Figura 3.29. Esta parte del algoritmo se encuentra dentro de un bucle *while* que garantiza la existencia de una coordenada, la cual es utilizada como punto meta en los algoritmos de obtención de ruta óptima (Capítulo 4). La estructura *case* controla el obtener o no nuevos valores para la

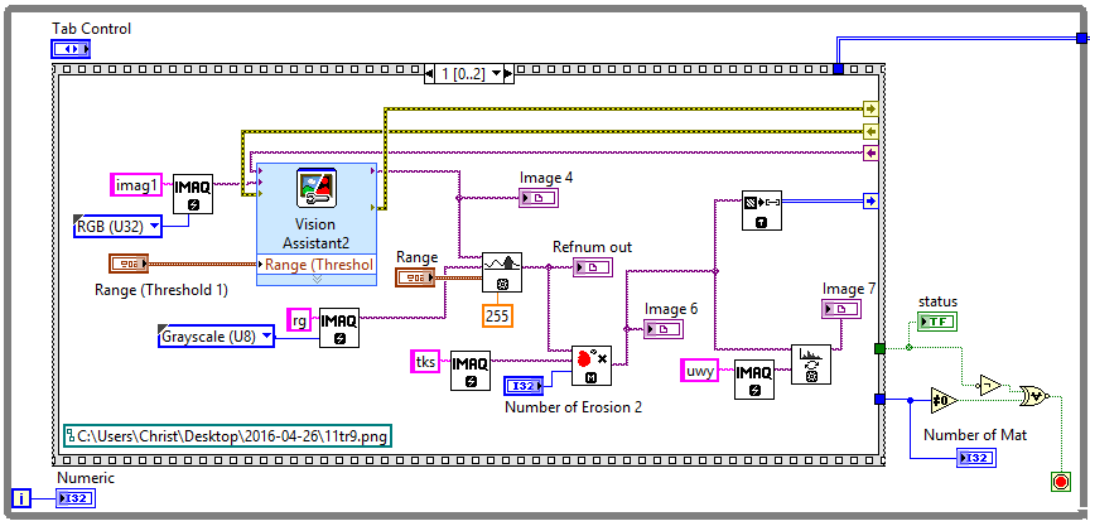
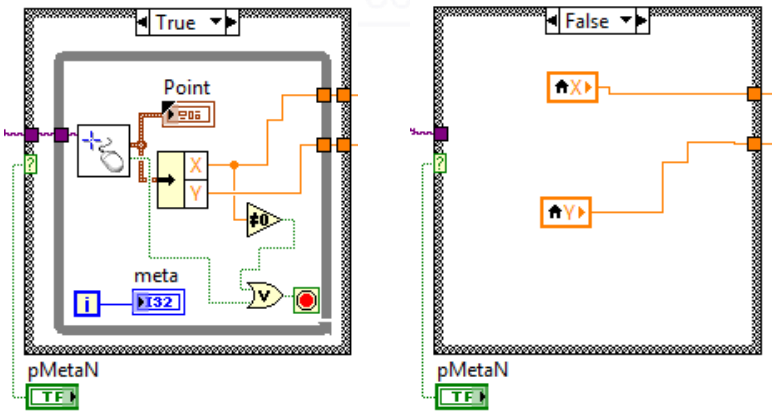


Figura 3.28: Diagrama de Bloques de la mejora de la imagen del entorno de DaNI

posición final de DaNI, lo que se realiza cuando en valor booleano de su selector es *true*, caso contrario se mantiene valores anteriores, en caso de existir, de la posición final de DaNI.



(a) Obtener nueva meta (*true*) (b) Obtener nueva meta (*false*)

Figura 3.29: Diagrama de Bloques para obtener la posición final deseada de DaNI

Generar la ruta óptima para el desplazamiento DaNI

La obtención de la ruta óptima se realiza mediante los algoritmos presentados en el Capítulo 4, desarrollados en MatLab. Los algoritmos requiere como parámetros de entrada una imagen binaria, la posición inicial y final de DaNI. De su ejecución se obtiene como resultado los valores correspondientes a las coordenadas de la ruta óptima si esta existe o caso contrario se recibe un cero. La Figura

3.30 presenta el bloque de LabVIEW y programación necesaria para llamar a ejecución los algoritmos desarrollados en MatLab.

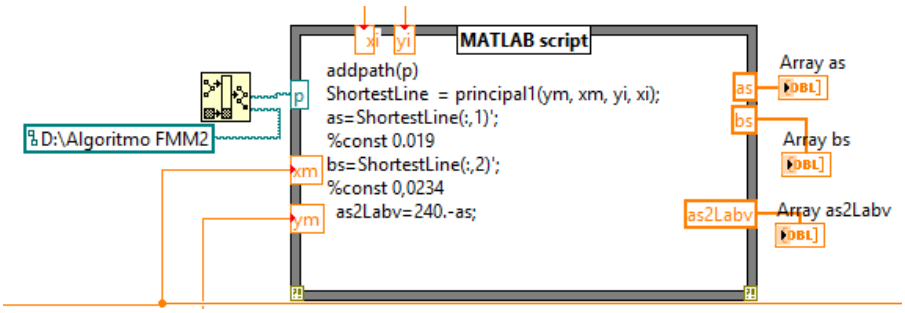


Figura 3.30: Ejecución del algoritmo de generación de ruta óptima

Desplazamiento de DaNI sobre la ruta óptima

El desplazamiento de DaNI se realiza de acuerdo a las coordenadas dadas por la ruta óptima y su orientación inicial. Luego, mediante la actuación de los motores de DaNI y enconders se controla su desplazamiento. Antes de proceder a iniciar el desplazamiento, se simplifica el conjunto de coordenadas para la ruta óptima tomando los valores cada 10 o 20 saltos (este valor se modificable mediante un botón de control). De este modo se reduce tiempo de procesamiento y además DaNI tiene mayor libertad de movimiento, ya que este no se encontraría limitado solamente a moverse entre píxeles adyacentes. La Figura 3.31 presenta la los bloques necesarios para realizar la simplificación de la ruta óptima.

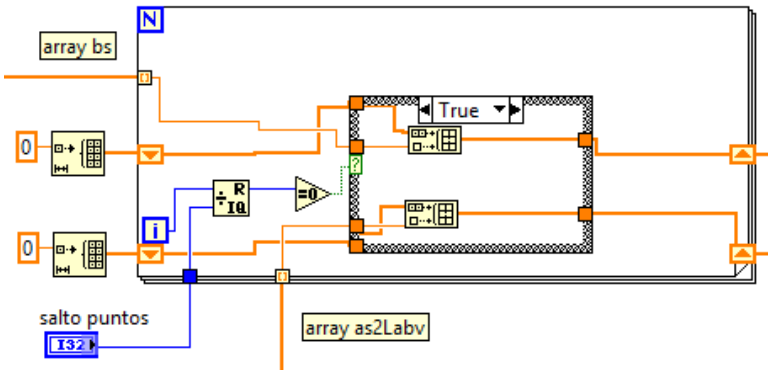


Figura 3.31: Simplificación de la ruta óptima

Teniendo los nuevos valores de la ruta óptima simplificada, se procede a calcular los valores de distancia y dirección entre coordenadas adyacentes comenzando desde el punto inicio hasta el punto meta de DaNI. La Figura 3.32 presenta la

programación necesaria. Se obtiene como resultado dos arreglos que corresponden a los valores de distancia y dirección.

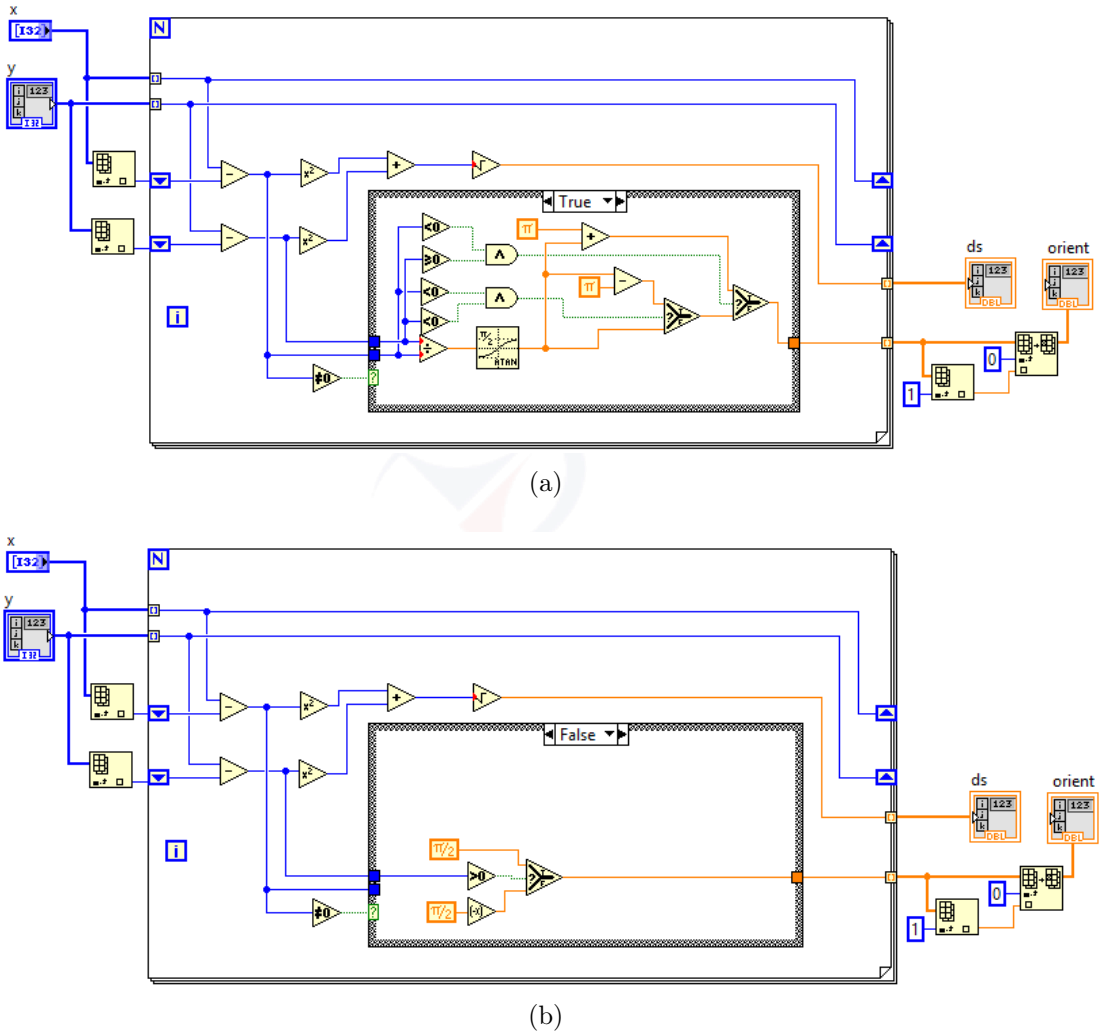


Figura 3.32: Obtención de distancia y dirección para el desplazamiento de DaNI

Para calcular la distancia desplazada de DaNI se hace uso de odometría. Esta técnica permite estimar la posición de un vehículo a partir del desplazamiento de sus ruedas (a partir de los datos de sus encoders). En los robot móviles diferenciales, como es el caso de DaNI, se dispone de encoders montados en cada motor para el conteo de revoluciones de las ruedas. Entonces, la odometría se realiza mediante simples ecuaciones geométricas para calcular la posición relativa del vehículo respecto a una posición inicial conocida [35].

Suponiendo que para un intervalo de muestreo I los encoders de las ruedas izquierda y derecha muestran un incremento de pulsos de N_L y N_R respectivamente.

Suponiendo además que,

$$c_m = \pi D_n / n C_e \tag{3.7}$$

donde:

c_m: Factor de conversión que convierte los pulsos del encoder a desplazamiento lineal de la rueda

D_n: Diámetro nominal de la rueda (en mm)

C_e: Resolución del encoder (en Pulsos Por Revolución)

n: Relación de engranaje del engrane de reducción entre el motor (donde está acoplado el encoder) y la rueda motriz

Podemos calcular el incremento de distancia viajada para la rueda izquierda y derecha, $\Delta U_{L,i}$ y $\Delta U_{R,i}$, de acuerdo a la Ecuación 3.8 y el incremento de desplazamiento lineal del punto central C del robot, denotado ΔU_i, de acuerdo a la Ecuación 3.9 [35].

$$\Delta U_{L/R,i} = c_m \, N_{L/R,i} \tag{3.8}$$

$$\Delta U_i = (\Delta U_R + \Delta U_L)/2 \tag{3.9}$$

Adicionalmente, podemos calcular la variación de la orientación de robot mediante la Ecuación 3.10 [35].

$$\Delta \Theta_i = (\Delta U_R - \Delta U_L)/b \tag{3.10}$$

donde "b" es la base del robot (distancia entre los dos puntos de contacto de las ruedas con el piso).

Finalmente, la nueva orientación relativa Θ_i se calcula mediante la Ecuación 3.11 [35].

$$\Theta_i = \Theta_{i-1} + \Delta \Theta_i \tag{3.11}$$

De acuerdo a las especificaciones de DaNI y su factor de relación de transmisión n=2, se genera el subVI denominado odometría. Su programación se observa en la Figura 3.33.

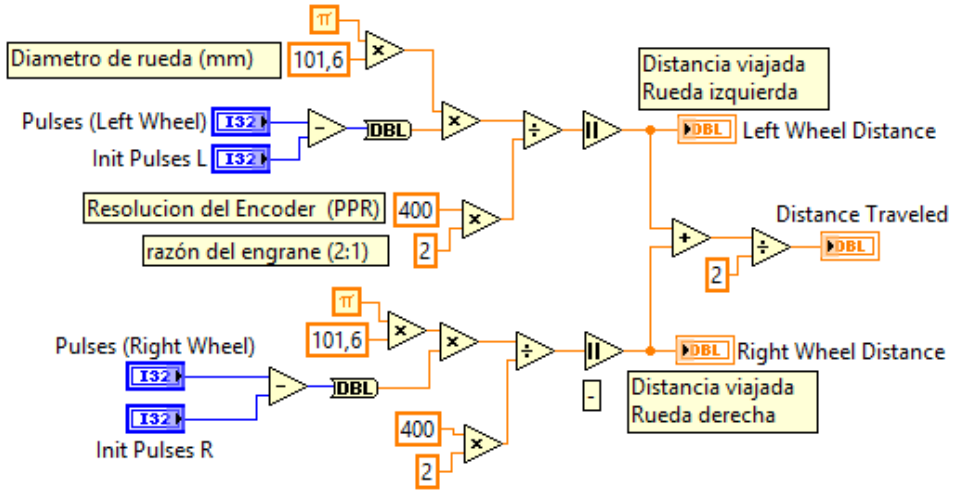


Figura 3.33: Algoritmo de odometría para estimar en desplazamiento de DaNI

El ejecución de los desplazamiento de DaNI se ha realizado en dos partes como son giros (Figura 3.34) y desplazamiento lineal (Figura 3.35). La programación para los giros se basa esencialmente en rotar a DaNI sobre su propio eje arcos correspondientes a la diferencia de dirección entre dos puntos consecutivos del ruta óptima simplificada. El desplazamiento lineal de DaNI se realiza mediante las distancias obtenidas del los puntos de la ruta óptima simplificada multiplicados por un factor de escala. El factor de escala relaciona un píxel de la imagen con su valor longitudinal equivalente a la escena real. Para obtener este valor se requieren las características de la cámara empleada y calcular su campo de visión (FOV, por sus siglas en inglés). En el caso del UGV DaNI se estableció un valor para la escala de 1cm por píxel, teniendo como resultado un escenario de 320cm $\times$ 240cm. Por tanto, se requiere calcular la altura correspondiente de la cámara para cumplir las condiciones de escala y FOV deseadas.

El FOV (Figura 3.36) de un sistema óptico generalmente se expresa como el tamaño angular(w) de un objeto visto desde la pupila de entrada (lente). También suele expresarse en medida plana (diagonal o lado de la escena). Sí la distancia (L) del lente al sensor es mucho menor a la altura o distancia de trabajo (WD, por sus siglas en inglés) se puede aproximar L a la distancia focal ($L \approx f$) [12].

Entonces, para objetos distante y suponiendo una lente delgada se tiene la siguiente Ecuación 3.12.

$$\frac{FOV}{WD} = \frac{SS}{f} \tag{3.12}$$

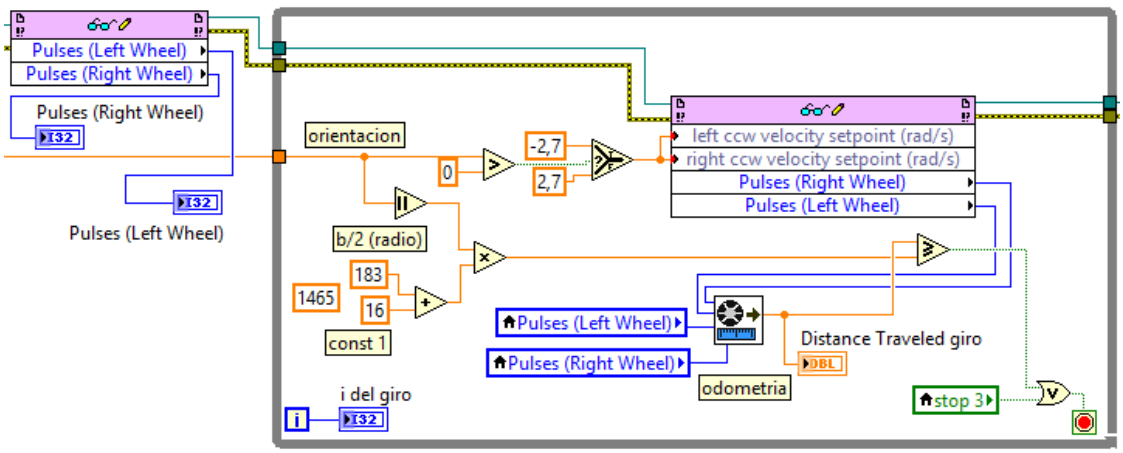


Figura 3.34: Algoritmo para ejecutar los giros de DaNI

Donde:

FOV: Campo de visión

WD: Distancia de trabajo

SS: Tamaño del sensor

f: Distancia focal

Con los datos de la cámara presentados en la Tabla 3.2 se procede a calcular su FOV.

Tipo de Sensor	**Tamaño[mm]**	**Distancia Focal [mm]**
CMOS BSI	4.54 × 3.42	4.6

Tabla 3.2: Características básicas de la cámara empleada en el desarrollo del UGV DaNI

Los cálculos se orientaron a determinar la WD dado que el FOV se estableció a valores de 320cm × 240cm, obteniendo como resultado:

$$WD = \frac{320[cm] \times 0{,}46[cm]}{0{,}454[cm]} \approx 324\,[cm] \tag{3.13}$$

Detección y evasión de obstáculos

La detección y evasión de obstáculos (que no se encontraban presentes al generar la ruta óptima) permite localizar nuevos objetos en el entorno de DaNI. DaNI ante la presencia de objetos en su trayectoria realiza acciones de parar su

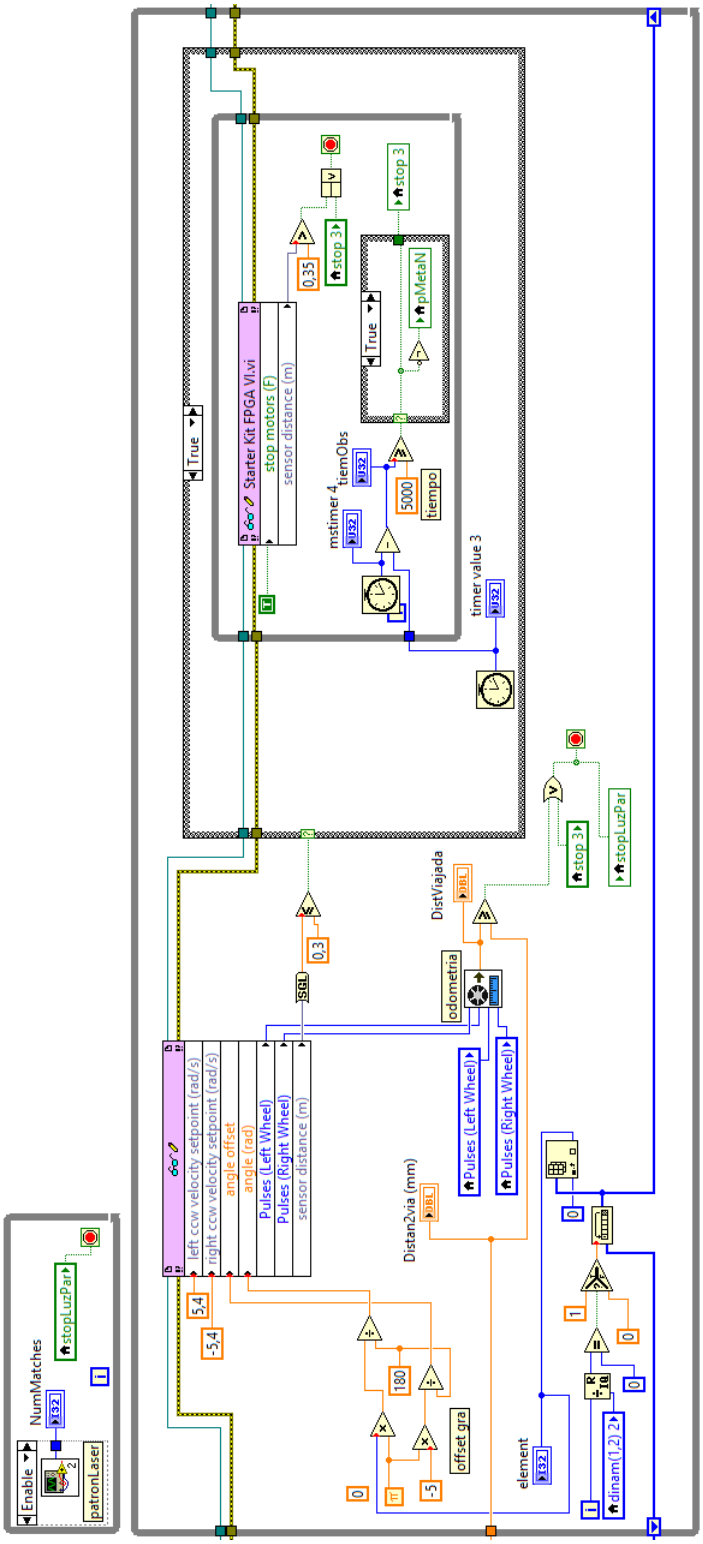


Figura 3.35: Algoritmo para ejecutar el desplazamiento lineal de DaNI

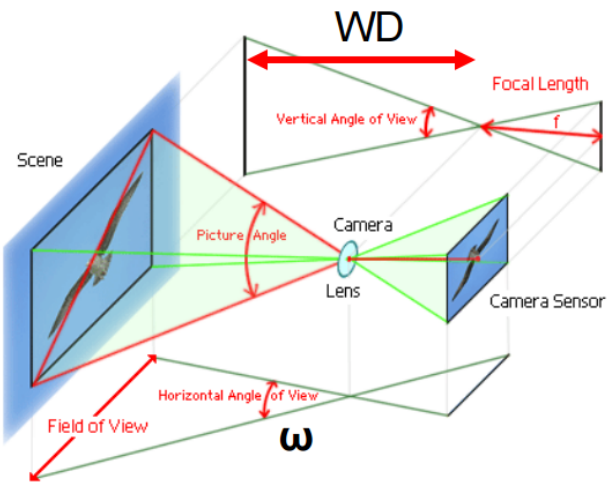


Figura 3.36: Ejemplificación del FOV de una cámara [12]

desplazamiento hasta que el objeto se retire. Si el objeto se encuentra obstruyendo el desplazamiento de DaNI por un periodo un tiempo t mayor a un valor fijo dado (5s), se calcula una nueva ruta. La detección de los objetos se realiza mediante iluminación estructurada en conjunto con visión artificial (Subsección 3.3.3) y/o mediante el sensor de ultrasonido (Subsubsección 3.4.1). Las dos técnicas implementadas para la detección de los obstáculos pueden ser usadas a la par o indistintamente según convenga. La Figura 3.37 presenta la programación para detectar un obstáculo y las acciones a realizar. Si se detecta un obstáculo mediante el sensor de ultrasonido (Figura 3.38), se ejecuta el caso *true* de la estructura *case*. Se ponen en *stop* los motores a espera de que el objeto se retire. Si el objeto permanezca por un tiempo mayor a $5s$ se realizar todo el proceso para generar una nueva ruta, con la particularidad que el punto inicio es donde DaNI se detuvo al encontrarse con el obstáculo y el punto meta es mismo de la primera ruta. Para el caso *false*, se realiza la detección de obstáculos mediante el análisis del patrón de iluminación estructurada. Si existe algún obstáculo (Figura 3.39), se procede de forma similar al caso *true* dado por el sensor de ultrasonido. Caso contrario DaNI se desplaza con normalidad (Figura 3.40).

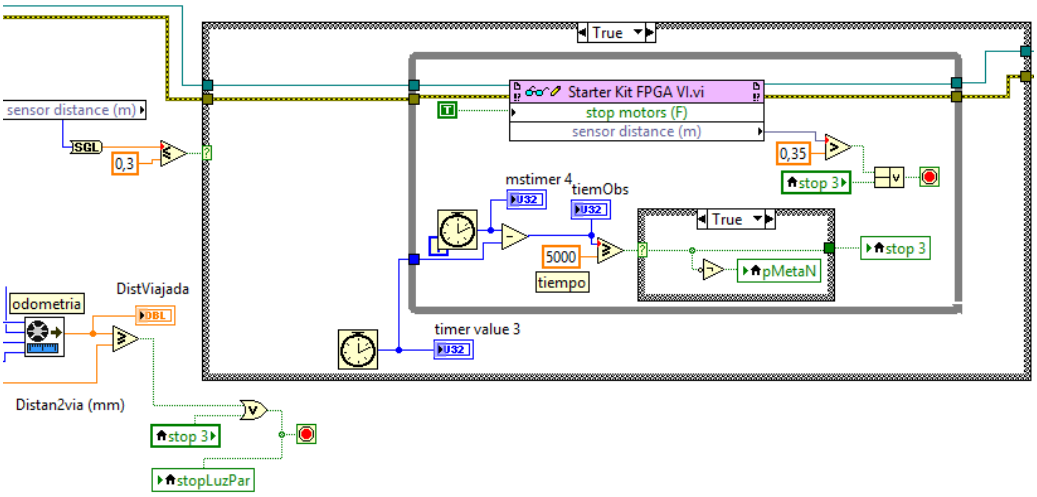


Figura 3.37: Algoritmo para detección y evasión de nuevos obstáculos

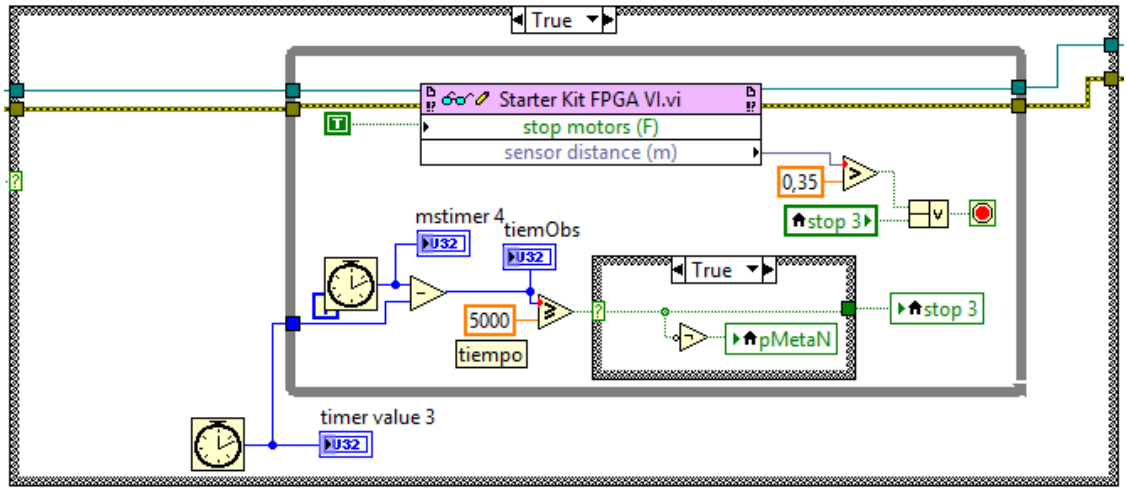


Figura 3.38: Detección de obstáculos mediante sensor de ultrasonido

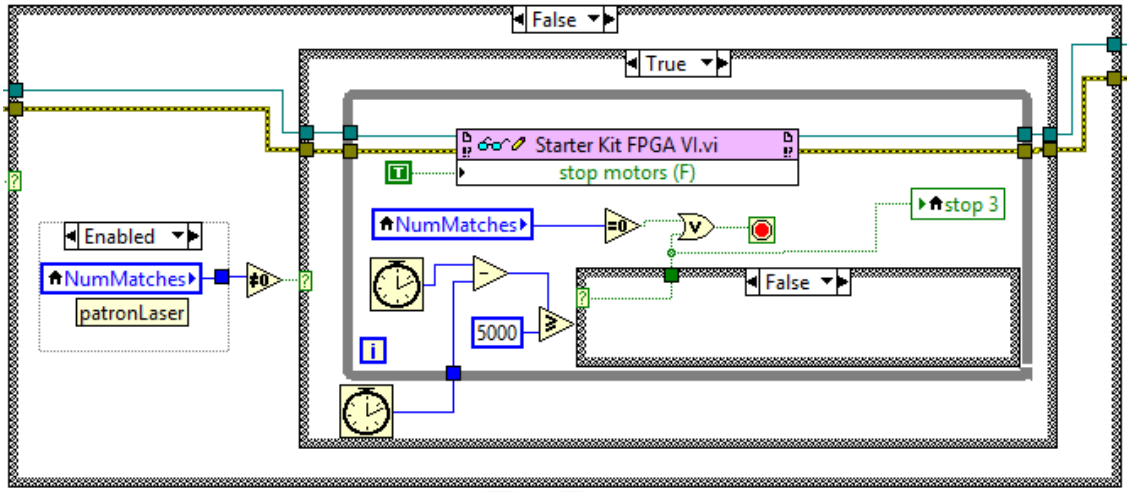


Figura 3.39: Detección de obstáculos mediante patrón de iluminación estructurada

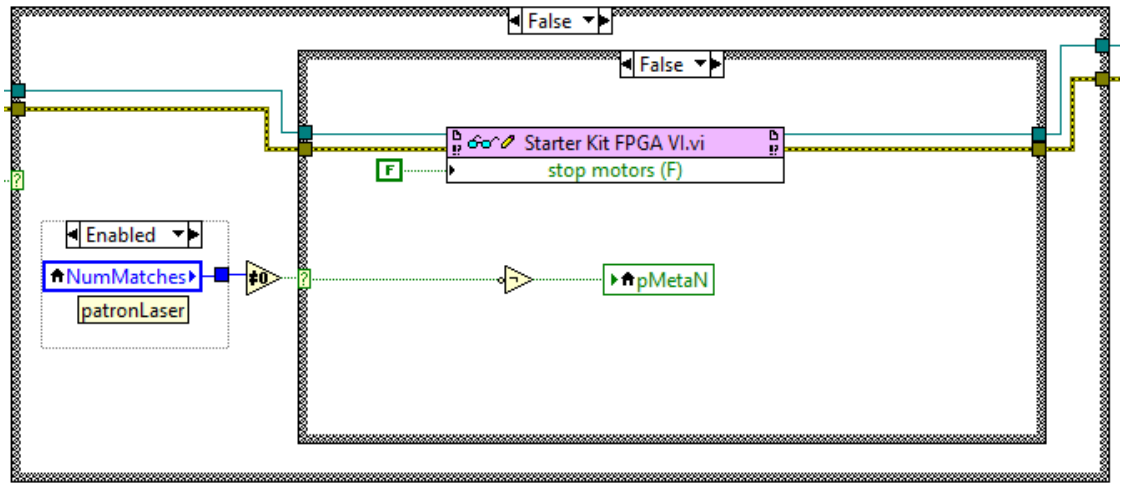


Figura 3.40: Detección de obstáculos. Caso *false* para el sensor de ultrasonido y el patrón de iluminación estructurada

Capítulo 4

Algoritmos para búsqueda de rutas de navegación

4.1. Introducción

La planificación de trayectorias es un campo activo de investigación que desenvuelve un papel fundamental en aplicaciones relacionadas con la robótica, aéreo navegación, industria automotriz, equipos médicos, plantas nucleares, etc. Hoy en día, existen ya diversos métodos de planificación de trayectorias capaces de generar rutas considerando una gran variedad de restricciones. No obstante, la gran mayoría de estos métodos no reúnen cualidades que demandan los sistemas reales, que están rigurosamente sesgados a la necesidad de una solución óptima: suavidad, eficiencia y bajo coste computacional. En consecuencia, la comunidad científica continua con la búsqueda de métodos que cumplan con estas características, enfocados a propósitos más generales y viables en sistemas reales.

Aunque existen variedad de métodos de planificación, este trabajo de titulación solamente se enfocara en los métodos geométricos. Los métodos geométricos son aquellos que tienen en cuenta únicamente la geometría del espacio para llevar a cabo la planificación, omitiendo el modelo cinemático o dinámico del robot . En virtud de los diferentes métodos geométricos existentes, La Valle [36–38] propone la clasificación de los mismos en las siguientes categorías:

- *Métodos combinatorios*: Son aquellos algoritmos que realizan la planificación de rutas mediante la construcción de estructuras de datos, capturando toda la información requerida en la planificación. El grupo de algoritmos más popular, representando a ésta categoría, son los basados en *mapas de rutas*; cuya función es buscar la ruta más corta mediante la interconexion de subrutas previamente calculadas. Un derivado de este grupo son los algo-

ritmos basados en *grafos de visibilidad* [39]. Estos algoritmos son aplicados principalmente a espacios bidimensionales cuyos obstáculos presentan configuraciones poligonales. Como se muestra en la Figura 4.1, cada subruta se origina de la intersección entre los vértices de cada polígono, el punto de inicio y el punto meta. Si cada subruta creada no intersecta los obstáculos, es considerada como un posible candidato para conformar la ruta final. Bajo este contexto, el algoritmo no encuentra una ruta óptima (si existe) sino una solución admisible [40].

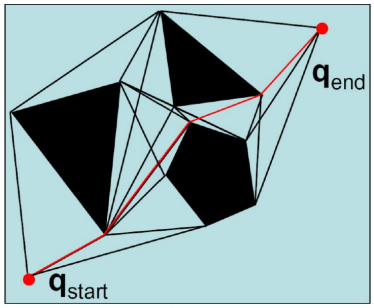


Figura 4.1: Grafo de Visibilidad [13]

- *Métodos basados en muestreo*: Este es otro enfoque muy popular en el campo de la planificación de trayectorias. Trabajan conjuntamente con algoritmos de detección de colisiones. Básicamente, la exploración de rutas se realiza mediante búsquedas incrementales en el espacio. El algoritmo más habitual en esta categoría es el *Rapidly Exploring Random Tree* (*RRT*) [41, 42] conjutamente con sus variantes. El RRT construye incrementalmente árboles de búsqueda para la exploración rápida y uniforme del espacio. Particularmente, se emplea en problemas que involucran restricciones diferenciales, tales como los robots no holonómicos [43]. La propiedad mas destacada de RRTs es su cobertura uniforme de espacios no convexos [42]. Sin embargo, debido a su propiedad estocástica, genera rutas muy pobres que carecen de propiedades de gran relevancia en la robótica móvil, como ser suaves, seguras y óptimas.

- *Métodos basados en espacios discretos*: Ciertas literaturas consideran que esta categoría podría ser parte de los métodos basados en muestreo. Sin embargo, existen diferencias distinguibles suficientes entre estas dos categorías que sustentan su división. Estos algoritmos antes de planificar, discretizan el espacio de interés, ya sea en grafos o rejillas compuestas por celdas o nodos, y acto seguido asignan una conectividad entre elementos. Generalmente , la

conectividad son costes asignados a cada celda para poder desplazarse entre ellas. La asignación de costes son establecidos en función de los criterios de cada algoritmo. El algoritmo A Estrella (A^*) es un algoritmo de naturaleza discreta que selecciona posibles nodos segun el criterio de mínima distancia (mejor heurística); si encuentra un nodo ocupado por un obstáculo este nodo se descarta (criterio de admisibilidad) [44]. A^* es una extensión del algoritmo Dijkstra, que provee un mejor rendimiento, respecto a tiempo, al usar heurística [45]. Finalmente, el método *Fast Marching Method* (FMM) se basa en un entorno discretizado como una malla (triangular o cuadrada). Los costes de llegada a cada celda se asigna en base a la expansión de una onda, y mediante esto se genera la trayectoria [46].

En el presente trabajo de titulación, las categorías a desarrollar son los *métodos basados en muestreo* y los *métodos basados en espacios*. Para los métodos basados en muestreo se tomará como referencia los algoritmos RRT y Bidirectional Exploring Random Tree ($B\text{-}RRT$). Por su parte, en la categoría métodos basados en espacios discretos estarán los algoritmos A^*, FMM y Fast Marching Square (FMM^2).

Todos estos algoritmos serán implementados en el robot DaNI y probados en un entorno real para comprobar su eficiencia ante complicaciones que no son contemplados en la teoría [47]. Cada uno de los algoritmos son descritos pormenorizadamente en las siguientes secciones, conjuntamente con su formulación matemática como base teórica para su implementación.

4.2. Rapidly exploring random tree

RRT es un algoritmo diseñado para buscar de manera eficiente rutas en mapas de alta dimensión mediante la construcción de arboles al azar. Aunque originalmente fue diseñado para planear el movimiento de un brazo humano, en un entorno gráfico de tareas de manipulación, es ampliamente utilizado en la planificación de trayectorias. Puede trabajar en distintos escenarios gracias a sus virtudes de adaptación a sistemas con restricciones cinemáticas y dinámicas [48]. La nomenclatura usada por este algoritmo son:

- *nodo*: Cada punto que conforma el árbol.
- *enlace*: Segmento de linea que une dos nodos.

- T: Matriz de nodos del árbol
- C_{libre}: Espacio libre de colisiones
- $q_{inicial}$: Es el punto de partida del robot (coordenadas x e y)
- q_{meta}: Es la posición final a alcanzar por el robot
- $q_{aleatorio}$: Es un punto aleatorio que se genera dentro del algoritmo, cuya premisa es proveer la dirección de expansión del próximo enlace, entre $q_{cercano}$ a q_{nuevo}
- $q_{cercano}$: Es el nodo más cercano a $q_{aleatorio}$ de entre todos los demás nodos existentes del árbol
- q_{nuevo}: Es el nuevo nodo a añadir al árbol. Esta ubicado a una magnitud y dirección dados por e_{enlace} y $q_{aleatorio}$ respectivamente
- l_{enlace}: Es la longitud de expansión de cada nuevo enlace, desde $q_{cercano}$ a q_{nuevo}

4.2.1. Funcionamiento

El árbol T se construye de forma incremental desde el punto de partida q_{inicio} a q_{meta}, a partir de muestras aleatorias $q_{aleatorio}$ extraídas del espacio de búsqueda. Todo este proceso se describe a grandes rasgos en el Código 4.1. Primero, se establece el nodo inicial de expansión del árbol, q_{inicio}. Segundo, se genera un $q_{aleatorio}$ a través de la función *GenerarAleatorio()*, como se muestra en la Figura 4.2. Tercero, se calcula el $q_{cercano}$ a $q_{aleatorio}$. Si es la primera iteración, $q_{cercano}$ es igual a q_{inicio}. Cuarto, calcular la dirección descrita entre $q_{cercano}$ a $q_{aleatorio}$ con *ObtenerDireccion()*, para determinar la dirección de expansión del árbol T. Quinto, se genera el q_{nuevo} con la función *Extender()* y se comprueba que este no haya sido creado previamente. Finalmente, se agrega el nodo creado al árbol T, para llevar una bitácora de estos.

El algoritmo RRT finalizará su ejecución siempre y cuando el nodo q_{nuevo} diste de la meta un radio menor o igual a la prolongación del enlace (l_{enlace}). Su propiedad más sobresaliente es que está intrínsecamente sesgada a crecer hacia las grandes áreas inexploradas del mapa (C_{libre}), pues en las zonas libres existe mayor posibilidad de encontrar puntos aleatorios factibles. Esto puede comprenderse mejor observando el crecimiento del árbol en diferentes entornos.

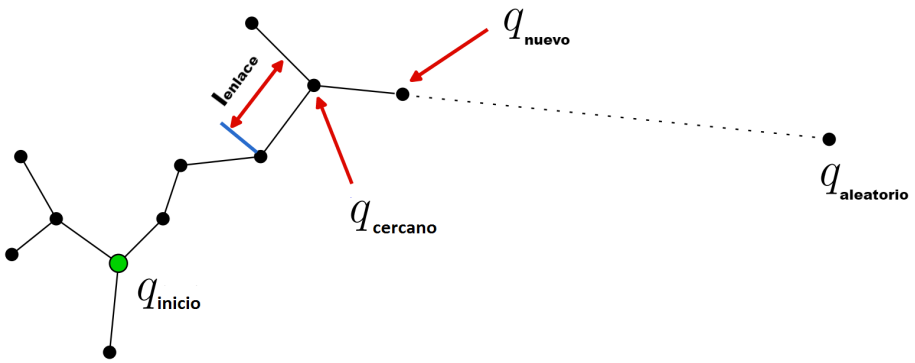


Figura 4.2: Algoritmo RRT: Generación de Nodos [14]

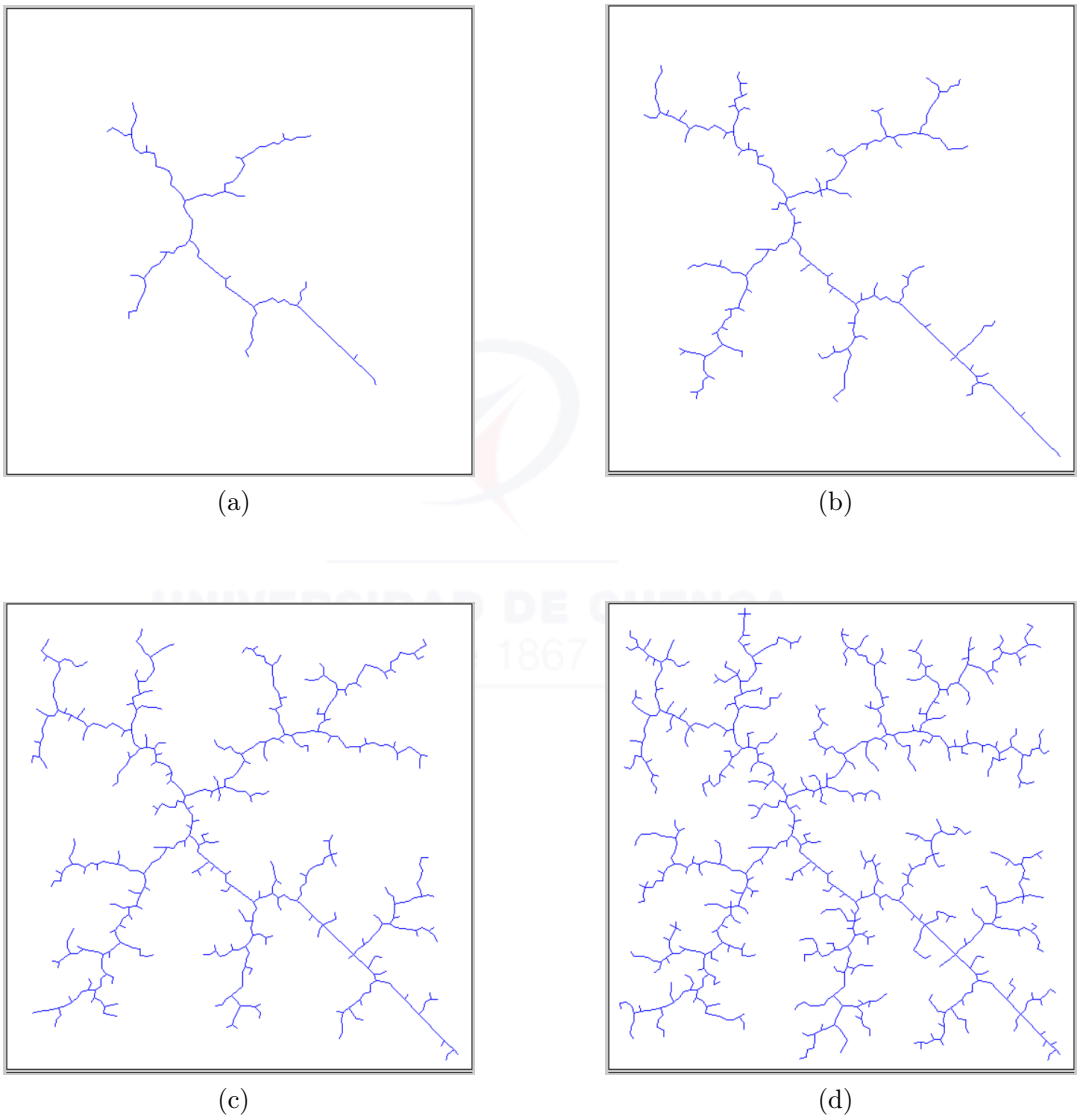
Algoritmo 4.1: Pseudocódigo Algoritmo RRT

```
T[][]= qinicio;

qinicio=qnuevo;
while distancia(qnuevo,qmeta)>lenlace

    qaleatorio= GenerarAleatorio(imagen);

    qcercano=NodoMasCercano(Tree,qaleatorio);

    direccion= ObtenerDireccion(qcercano,qaleatorio);

    qnuevo=Calcular(qcercano,direccion,lenlace);

    Extender(qcernano, qnuevo);

    Tree=AgregarNodo(qnuevo);

end
```

4.2.2. Simulaciones

Para las simulaciones se han escogido dos entornos: uno con obstáculos y otro sin obstáculos. Para el caso de un entorno con obstáculos, la Figura 4.3 ilustra el desarrollo secuencial del algoritmo en 4 instantes distintos de tiempo. Por la ausencia de obstáculos, el árbol presenta una distribución uniforme en todas direcciones. La carencia de predominio entre ramas origina árboles más densos, como se ilustra en la Figura 4.3d. Naturalmente, árboles más densos originan un

mayor coste de procesamiento. Sin embargo, es posible realizar ligeras manipulaciones al algoritmo para ahorrar procesamiento, priorizando la generación de ramificaciones con tendencia hacia la meta.



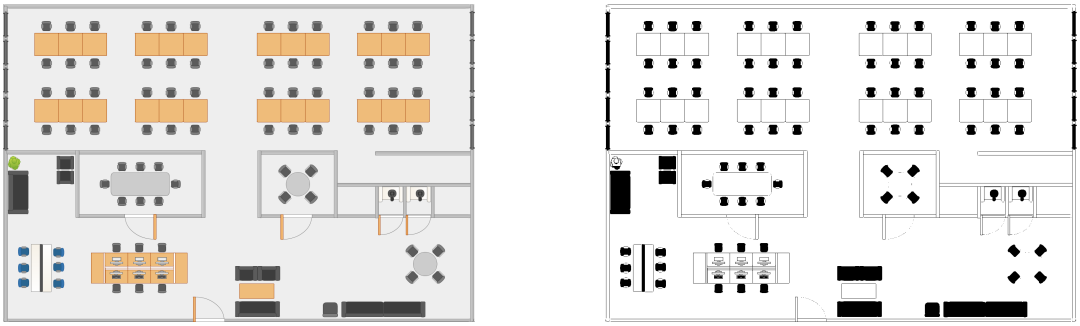
(a) (b)

(c) (d)

Figura 4.3: Convergencia del algoritmo RRT en un entorno C_{libre}

Análogamente, para el caso con obstáculos, la Figura 4.4 muestra dos figuras. La Figura 4.4a representa la imagen con sus características naturales y la Figura 4.4b la imagen en formato bmp (mapa de bits monocromático). Cabe señalar, que la imagen procesada por el algoritmo es el de la Figura 4.4b.

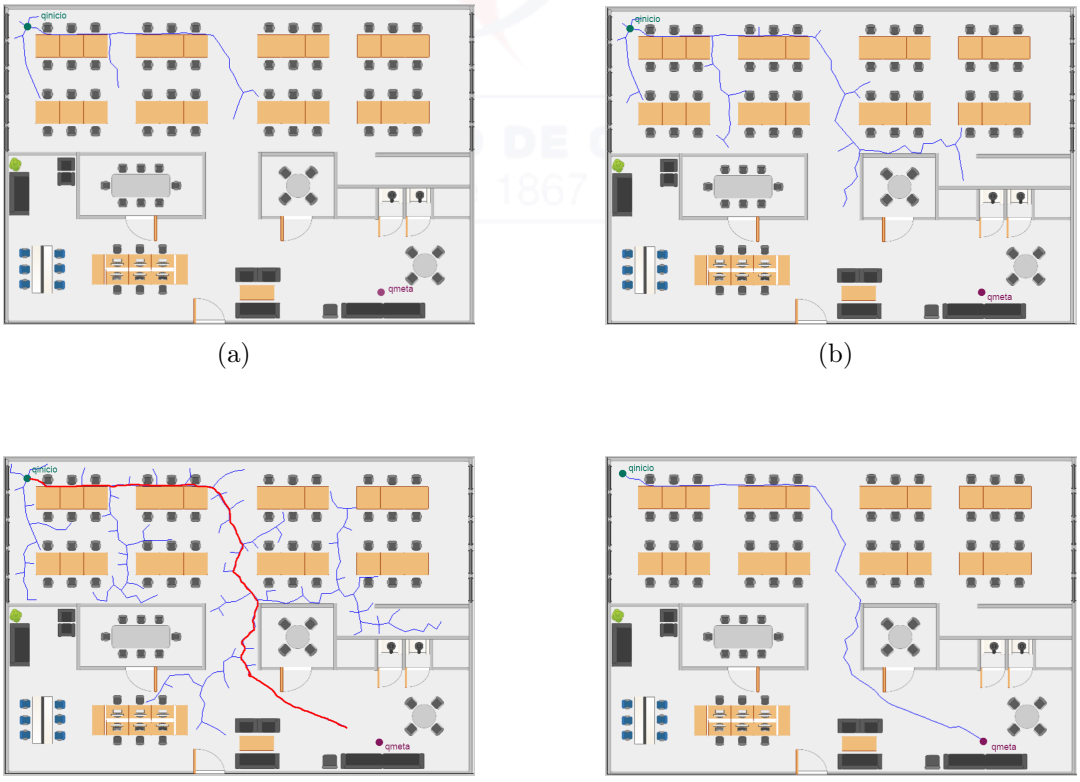
Para las simulaciones de la Figura 4.5, se ha otorgado cierto nivel de prioridad a las rutas con dirección hacia la meta. El nodo de inicio es el nodo etiquetado de color verde y el nodo meta es aquel etiquetado de color violeta. Gracias a la



(a) (b)

Figura 4.4: Entorno de evaluación con obstáculos

prioridad agregada, se puede notar claramente como el árbol intenta sustentar la condición a través de ramificaciones apresuradas hacia q_{meta}.



(a) (b)

(c) (d)

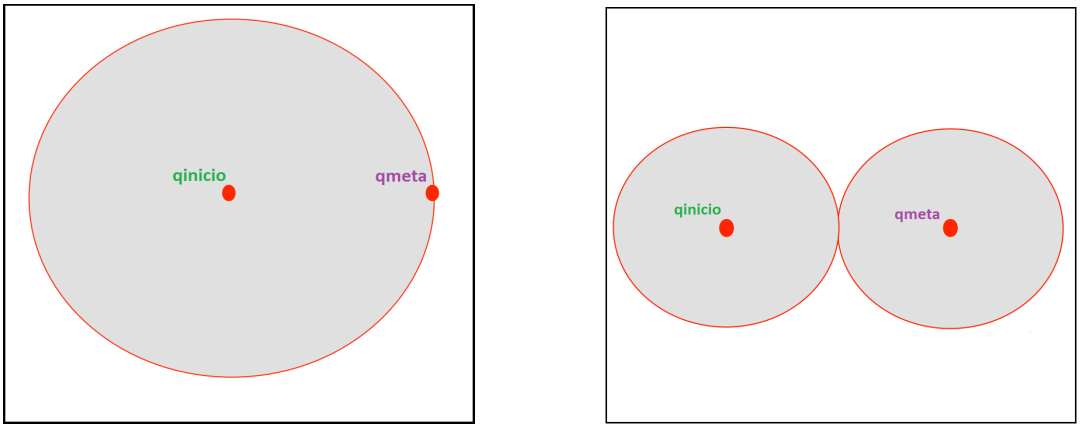
Figura 4.5: Convergencia del algoritmo RRT en un entorno con obstáculos

La Figura 4.5a es el resultado luego de 60 iteraciones, la Figura4.5b luego de 180 iteraciones, la Figura 4.5c luego de 450 iteraciones y la trayectoria final, Figura 4.5d, luego de 500 iteraciones.

En conclusión, RRT ofrece homogeneidad del espacio explorado respecto a otros algoritmos. Su naturaleza exige, en un inicio, avanzar con mayor ávidez hacia zonas inexploradas. Como su estructura lo muestra, en el origen se observa la existencia de una gran densidad de ramas. Sin embargo, conforme el árbol va creciendo y cubriendo el espacio C_{libre}, aumenta el número de ramas que emergen por doquier.

4.3. Bidirectional rapidly exploring random tree

El algoritmo RRT ha demostrado ser probabilísticamente completo y computacionalmente eficiente en la planificación de trayectorias. La eficiencia de esta técnica radica en la selección adecuada de parámetros tales como: el paso de búsqueda, la definición de obstáculos y la configuración del espacio. Generalmente, no es necesario establecer pasos de búsqueda extremadamente pequeños para cubrir la exploración de pasajes estrechos; más bien, el paso de búsqueda puede tomar valores próximos o iguales a la distancia más corta entre dos obstáculos con el fin de reducir el número de iteraciones. La Figura 4.6 rescata la idea general usada por los algoritmos RRT básicos y bidireccionales. La Figura 4.6a representa el volumen barrido por el algoritmo RRT y la Figura 4.6a el volumen barrido por los algoritmos bidireccionales.



(a) (b)

Figura 4.6: Comparación entre los algoritmo RRT y bidirectional rapidly exploring random tree

El barrido simultáneo de árboles, cuyas raices son q_{inicio} y q_{meta}, evita la exploración fina de todo el entorno, ahorrando grandes cantidades de procesamiento. Variantes bidireccionales del algoritmo [49] son aplicados en entornos de difícil

acceso, tales como pasillos estrechos o espacios multidimensionales con altas concentraciones de obstáculos. Las variantes bidireccionales más populares son: el algoritmo B-RRT, RRT-Connect [48], RRT-Met [50] y RRT Óptimo [51]. Este apartado está enfocado solamente al algoritmo general B-RRT. La nomenclatura usada por este algoritmo es similar a RRT, con el agregado adicional de etiquetas al par de árboles generados, T_a y T_b.

4.3.1. Funcionamiento

La idea principal de B-RRT es construir un par de arboles T_a y T_b cuyo nacimiento tenga origen en los puntos q_{inicio} y q_{meta} respectivamente. Los dos arboles crecen simultáneamente uno hacia el otro, permitiendo una mejora en la convergencia del algoritmo, asi como en el tiempo de computo, debido al menor numero de comprobaciones. La estructura general del algoritmo se presenta en

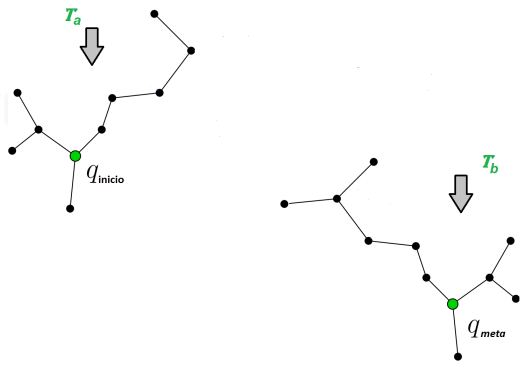
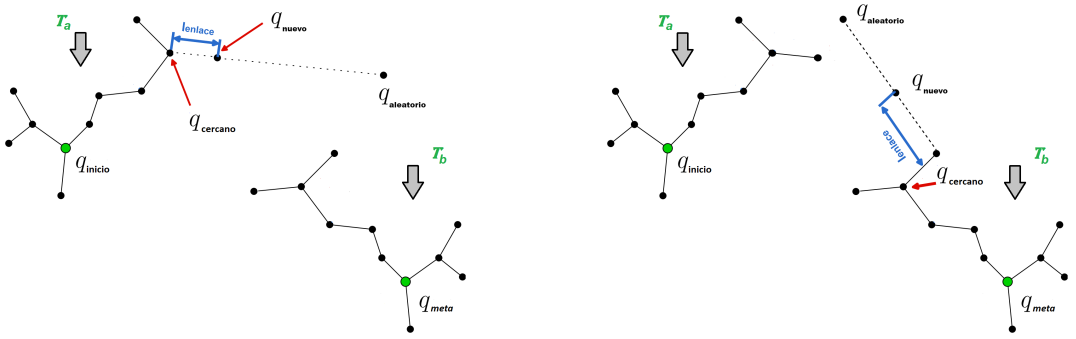


Figura 4.7: Estructura general del Algoritmo B-RRT

la Figura 4.7. En este algoritmo, T_a y T_b coexisten todo el tiempo mientras se encuentre una solución. Por lo tanto, en cada iteración uno de los arboles agregará una nueva rama, si es posible, en dirección al punto generado $q_{aleatorio}$, como muestran las Figuras 4.8a y 4.8b para los arboles T_a y T_b respectivamente.

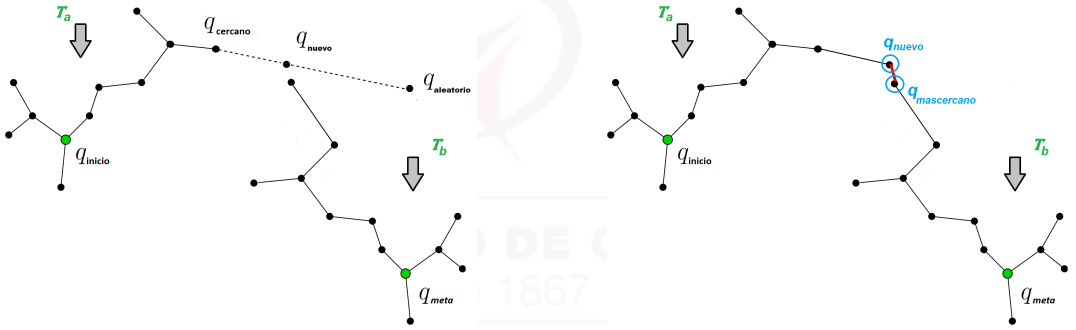
Todo el procedimiento anterior es repetido vez tras vez hasta obtener un escenario similar al de la Figura 4.9a. En esta instancia los arboles T_a y T_b se han prolongado lo suficiente; al punto que, el próximo q_{nuevo} a generar del árbol T_a dista una distancia menor a l_{enlace} del nodo perteneciente al árbol T_b. Este escenario finaliza la generación de nodos.

La finalización de generación de nodos es validada durante cada etapa de generación de arboles. La validación implica buscar en el árbol no generador del ultimo q_{nuevo} el nodo más cercano a q_{nuevo}, que diste una distancia menor o igual



(a) (b)

Figura 4.8: Algoritmo B-RRT: Generación de Nodos en T_a y T_b.



(a) (b)

Figura 4.9: Finalización de generación de Nodos en T_a y T_b.

a l_{enlace}. Para nuestro caso, el último q_{nuevo} es generado por el árbol T_a, Figura 4.9a, por lo tanto , la búsqueda del nodo más cercano se lleva a cabo en el arbol T_b, Figura 4.9b.

El resultado final del algoritmo de búsqueda B-RRT es presentado en la Figura 4.10. La trayectoria resaltada de color rojo describe la ruta encontrada para trasladarse desde q_{inicio} a q_{meta}. Cabe destacar que la ruta encontrada no es la óptima; sin embargo, el algoritmo en cuanto a tiempo de procesamiento responde favorablemente, y además su implementación es sencilla.

A continuación se presenta una descripción a grandes rasgos de la implementación del algoritmo B-RRT. Por motivos de eficiencia se ha dividido el código en dos partes. La primera parte, Código 4.2, se encarga tanto del intercambio entre arboles para su expansión como del término de la ejecución del algoritmo. El intercambio se realiza a través de la función *Generacion*(), que se describirá más adelante, y la finalización se lleva a cabo mediante la variable *salir*. La variable *salir* se encarga de sensar el acercamiento que existe entre nodos de los

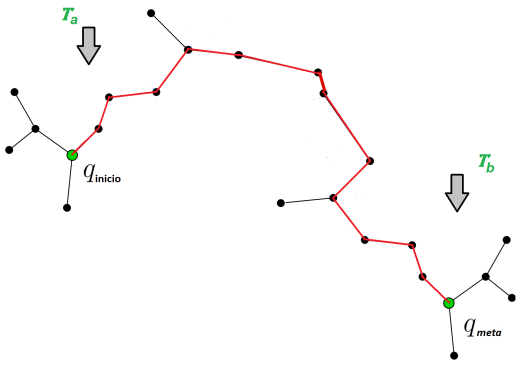


Figura 4.10: Trayectoria generada por el algoritmo B-RRT

dos árboles, T_a y T_b, y en base a ésta, la rutina principal decide finalizar o no la ejecución.

Algoritmo 4.2: Pseudocódigo Algoritmo B-RRT

```
Ta[0][0]= qinicio;

Tb[0][0]= qmeta;

lenlace;

salir=true;

while salir

    [Ta,salir]=Generacion(Ta, Tb, meta, lenlace,imagen);

    [Tb,salir]=Generacion((Tb, Ta, meta, lenlace,imagen);

end
```

La segunda parte es la función *Generacion*(), Código 4.3, encargada de realizar todo el proceso de expansión, validación y almacenamiento de nodos. Ésta función tiene integrada algunas funciones adicionales en comparación con el algoritmo RRT, sin embargo, la mayoría de ellas desempeñan el mismo papel. Tales son el caso de las funciones *GenerarAleatorio*(), *NodoMasCercano*(), *ObtenerDireccion*(), *Calcual*(), *Extender*() y *AgregarNodo*(); por tal motivo, no se detallarán pormenorizadamente dichas funciones. Se recomienda revisar el Codigo 4.1.

Algoritmo 4.3: Pseudocódigo de la función *Generación()* del algoritmo B-RRT

```
function[Arbol1,salir]=Generacion(Arbol1,Arbol2,meta,lenlace,imagen)

MaximosIntentos;
while NumeroIntentos≤MaximosIntentos

    qaleatorio= GenerarAleatorio(imagen);

    qcercano=NodoMasCercano(Arbol1,qaleatorio);

    direccion=ObtenerDireccion(qcercano, qaleatorio);

    qnuevo=Calcular(qcercano,direccion,lenlace);

    if VerificarExistencia(qnuevo, Arbol1,imagen)=true
       numeroIntentos= numeroIntentos+1;
       continue;
    end

    if UbicadoObstaculo(qnuevo)
      numeroIntentos= numeroIntentos+1;
      continue;
    end

    qcercano2=NodoMasCercano(Arbol2,qnuevo);

    if distancia(qcerncano2, qnuevo)≤lenlace
       salir=false;
       break;
    end

    Extender(qcercano, qnuevo);

    Arbol1= AgregarNodo(qnuevo);

    break;
end
```

Luego de haber generado el $q_{aleatorio}$ y encontrado el $q_{cercano}$, y en base a este par haber obtenido q_{nuevo}, se procede a validar o verificar su existencia en el árbol generador. Si q_{nuevo} ya existe, se reinicia la función *Generacion*() hasta encontrar un q_{nuevo} que no esté almacenado en la bitácora T_a o T_b, según sea el caso. Si

la variable *NumeroIntentos* supera al valor de la variable *MaximoIntentos*, la función *Generacion*() finalizará y devolverá el mando a la rutina principal. Una vez en la rutina principal se intercambiara el árbol a expandir y se llamará nuevamente a la función *Generacion*().

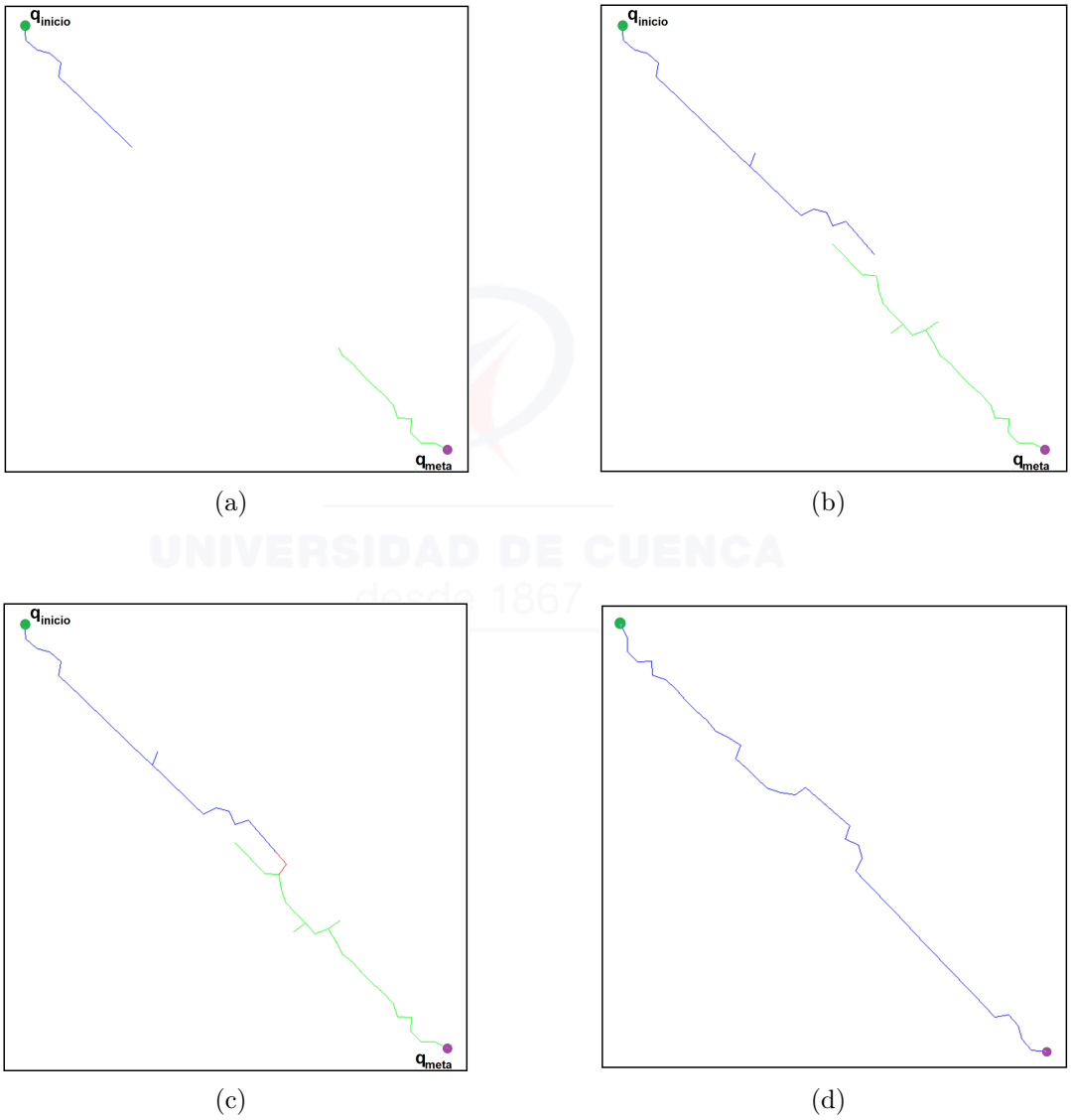


Figura 4.11: Convergencia del algoritmo B-RRT en un entorno sin obstáculos.

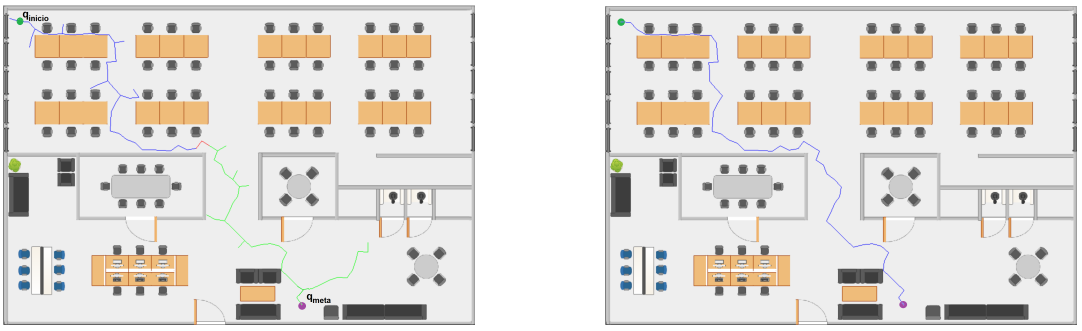
Otra característica de gran relevancia de la función *Generacion*() es la rutina *UbicadoObstaculo*(), cuya trabajo es verificar si el q_{nuevo} creado está ubicado sobre un obstáculo, de ser el caso, se pide generar un q_{nuevo} diferente. También se cuenta con un número máximo de intentos.

Cuando el nodo q_{nuevo} ha logrado pasar las dos validaciones anteriores, se comprueba si éste ha alcanzado ya la distancia mínima permitida (l_{enlace}) con algún

nodo del otro árbol. De ser así, la ruta ha sido encontrada y todo el algoritmo termina su ejecución. Sin embargo, si no existen nodos que cumplan el criterio de la mínima distancia, el algoritmo sigue su cause natural: extender la nueva rama desde $q_{cercano}$ a q_{nuevo} y almacenar este último. Esta concepción relativamente simple de los algoritmos RRT son aplicables a espacios multidimensionales. No obstante, la eficiencia de los algoritmos están altamente sujetos al rendimiento que estos proveen en sistemas reales ante colisiones. En las siguientes simulaciones se evaluará estos aspectos.

4.3.2. Simulaciones

De la misma manera que para el algoritmo RRT, los escenarios escogidos son dos: sin obstáculos y con obstáculos. El entorno con obstáculos es el de la Figura 4.4a. La simulación realizada en el entorno vació es mostrado en la Figura 4.11. En comparación con su similar RRT, el algoritmo $B-RRT$ origina menos ramificaciones y en consecuencia un tiempo de convergencia mucho menor, resaltando el claro liderazgo del algoritmo sobre su homólogo. A su vez, la evolución de los árboles en un entorno con obstáculos también es considerablemente reducida como efecto directo del crecimiento paralelo de árboles. La Figura 4.12 ilustra este hecho.



(a) (b)

Figura 4.12: Finalización de generación de Nodos en T_a y T_b.

En conclusión, los algoritmos RRT pueden incrementar su precisión simplemente al reducir el paso de búsqueda, sin embargo, esta acción provoca un tiempo de computo mucho mayor. Adicionalmente, se puede aplicar diferentes técnicas de pre-procesamiento para lograr trayectorias suaves y cercanas a la óptima [18], sin recurrir a la reducción del paso de búsqueda. La ventaja principal de los algo-

ritmos RRT es su velocidad de convergencia en entornos abiertos. No obstante, cuando el entorno esta infestado de obstáculos, el tiempo de procesamiento se incrementa considerablemente. Bajo este contexto, la estrategia es incrementar la cantidad de arboles dentro del entorno de búsqueda cuyo procesamiento se lo realice en paralelo.

4.4. A star

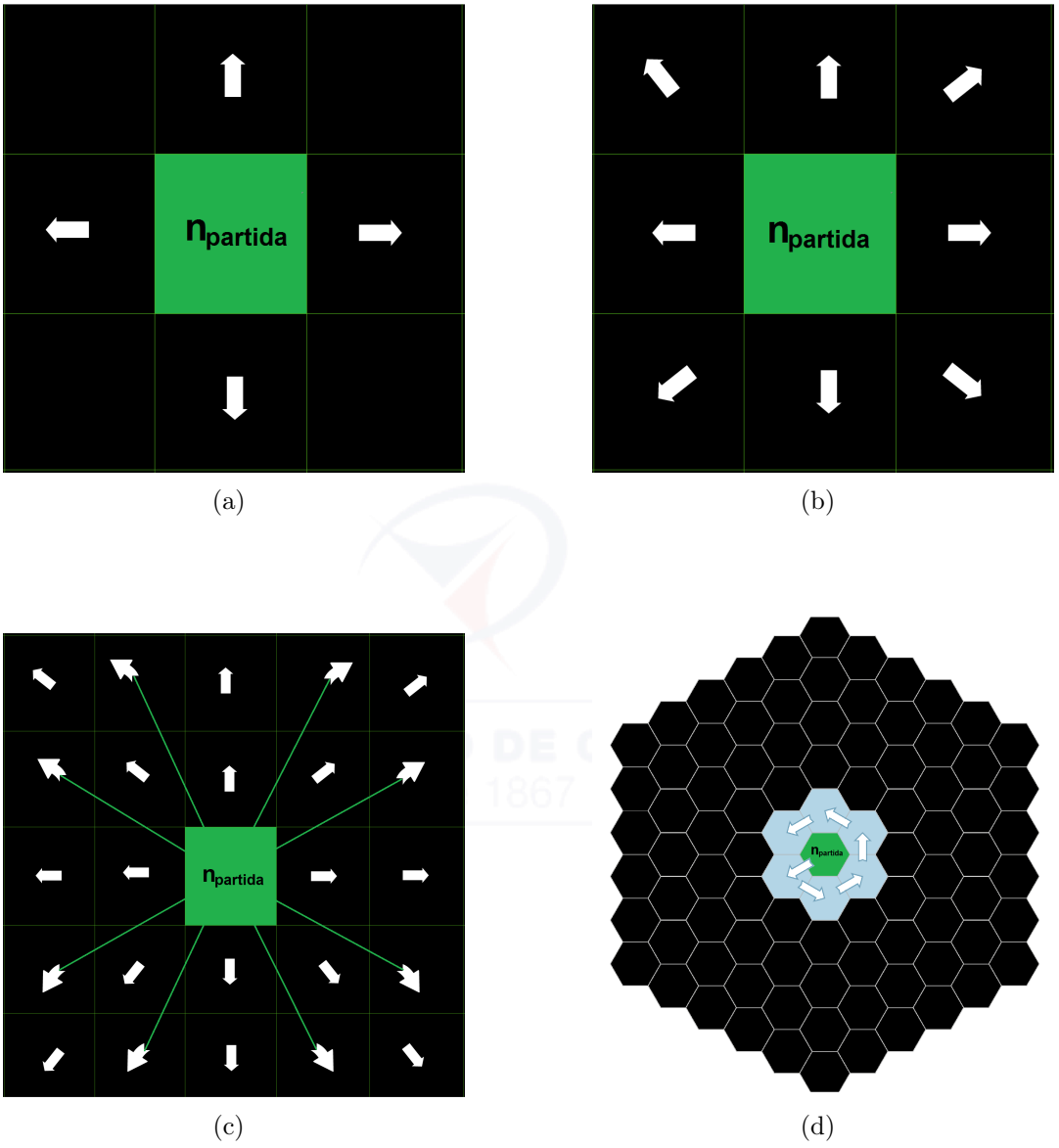
A^* es probablemente uno de los algoritmos de inteligencia artificial más conocidos. Su objetivo es encontrar la ruta más corta en mapas cuadriculados desde un nodo de partida a un nodo meta [52, 53]. Esta técnica usa propiedades tanto del algoritmo Dijkstra [54, 55] como el de Búsqueda Primero el Mejor (Best-First-Search) [56]. La combinación de búsqueda de costo uniforme y búsqueda heurística pura permiten calcular eficientemente soluciones óptimas [57]. La terminología estándar usada por el algoritmo A^* se muestra en la Ecuación 4.1

$$f(n) = g(n) + h(n) \tag{4.1}$$

donde $g(n)$ es el costo de alcance del nodo n, ponderado desde q_{inicio}; $h(n)$ es una estimación del costo de alcance de q_{meta}, ponderado desde el nodo n.

La Ecuación 4.1 ofrece maleabilidad gracias a su simple manipulación de parámetros. Por ejemplo:

- Si $h(n)$ es 0, $f(n) = g(n)$, y A^* se reduce al algoritmo Dijkstra, el cual garantiza encontrar la trayectoria más corta.
- Si $h(n)$ es siempre menor que el costo de traslado desde n hacia la meta, A^* expande más nodos ocasionando una convergencia más lenta.
- Si $h(n)$ es algunas veces más grande que el costo de n a la meta, A^* no garantiza encontrar el camino más corto, pero si su convergencia rápida.
- Si $h(n)$ es exactamente igual al costo de n a la meta, el algoritmo solamente tomará la mejor ruta y nunca se expandirá hacia otro lado, logrando una convergencia extremadamente rápida. Aunque es muy complicado hacer que esto suceda en la mayoría de casos, pero existen excepciones.
- Si $h(n)$ es muy grande relativo a $g(n)$, entonces $f(n) = h(n)$, y A^* se reduce al algoritmo de Búsqueda Primero el Mejor.



(a) (b)

(c) (d)

Figura 4.13: Matrices de conexión

La implementación del algoritmo A^* exige como pre-requisito el conocimiento de tres factores:

- Primero, saber como convertir o transformar el problema de interés a un mapa discreto y en base a esto determinar la matriz de conexión para la búsqueda de la ruta más corta. Para conveniencia nuestra, la discretización de la imagen es realizada fácilmente, en donde cada cuadricula o píxel representa un nodo y cada nodo un posible punto de movimiento. Los movimientos permitidos entre nodos describen la matriz de conexión, cuya geometría puede ser distintas configuraciones poligonales. El centro de cada matriz de

conexión se denomina nodo de partida, $n_{partida}$.

La Figura 4.13 muestra cuatro de las más usadas. Las Figuras 4.13a, 4.13b y 4.13c ilustran las matrices de conexión para movimientos en 4, 8 y cualquier dirección respectivamente. A su vez, la Figura 4.13d es una matriz de conexión hexagonal con 6 grados de libertad. Las tres primeras matrices se implementarán en este documento para determinar el más eficiente.

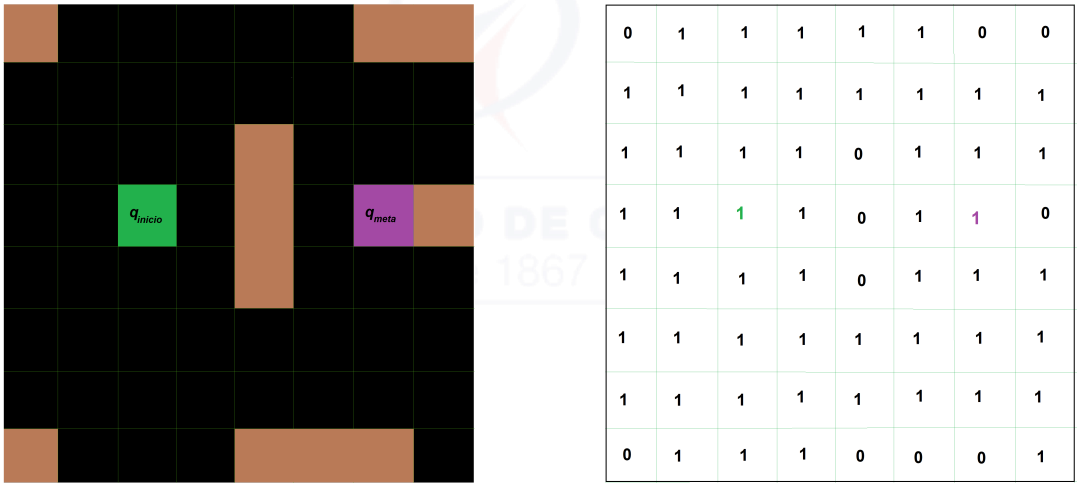
- Segundo, conocer una heurística admisible [58, 59] para el problema. Una heurística admisible es aquella que calcula el costo estimado del nodo n a la meta, y no la sobrestima. El uso de heurísticas admisibles en el algoritmo A^* garantiza una búsqueda rápida y óptima. Sin embargo, la determinación de dicha heurística [60] no es una tarea fácil, y en la mayoría de los casos resulta muy complejo su calculo [61]. Afortunadamente, para nuestro propósito, existen heurísticas eficientes y muy fáciles de implementar. Las heurísticas a usarse en este trabajo de titulación son heurísticas basadas en distancias. Las funciones heurísticas basadas en distancias más conocidas son: Distancia de Manhattan, Distancia Diagonal, Distancia Euclidiana y Distancia Euclidiana al Cuadrado. La diferencia entre una y otra radica esencialmente en su escenario de aplicación (configuraciones geométricas), ilustrados en la Figura 4.13. Estas cuatro técnicas serán implementadas con el ánimo de escoger la mejor.

- Tercero, saber o descubrir el costo requerido para viajar desde un nodo a otro. Comúnmente, el costo de desplazamiento entre nodos es igual a 1 para el movimiento vertical y horizontal, y 1.414 para el movimiento en diagonal; cuyo valor es consecuencia de la raíz cuadrada de dos. No obstante, para evitar el cálculo de raíces y el acarreo de decimales, y en consecuencia ahorrar tiempo de procesamiento, este trabajo de titulación usa los valores enteros de 10 y 14.

El lector debe entender que los tres factores antes mencionados son los requisitos básicos, pero suficientes, para la correcta implementación y manipulación del algoritmo. Cualquier variante del algoritmo da por sentado el conocimiento de los tres criterios. De entre los tres, la parte más delicada es determinar la adecuada heurística para el problema. En la sección 4.4.1 se detalla la evolución del algoritmo, así también los resultados de simulaciones.

4.4.1. Funcionamiento

La intensión de éste apartado no es la explicación teórica completa del algoritmo, por el contrario, solo presentar información coherente, amigable y suficiente para que el lector sea capaz de entenderlo y plasmarlo sin dificultad en los diferentes entornos de programación. Bajo este criterio y haciendo caso omiso a los tres pre-requisitos antes mencionados, se considera la Figura 4.14. La Figura 4.14a ilustra un entorno de búsqueda simplificado, conformado por obstáculos, el punto de partida y la meta; etiquetados con los colores café, verde y violeta respectivamente. Análogamente, la Figura 4.14b muestra el mismo escenario pero previamente aplicado procedimientos de discretización .



(a) (b)

Figura 4.14: Escenario de pruebas del algoritmo A^*

Una vez discretizada el área de búsqueda se seleccionada la matriz de conexión. Para nuestro propósito, se ha elegido la matriz de la Figura 4.13b, cuya representación discreta se muestra en la Figura 4.15. Hasta este punto, ya se ha cumplido con el primer pre-requisito.

Un paso adicional, inherente del proceso, es la clasificación de los nodos en dos bitácoras: *listaAbierta* y *listaCerrada*. La bitácora *listaAbierta* almacena nodos que pueden o no formar parte de la ruta a tomar. Por lo tanto, es una lista que necesita ser verificada. Contrariamente, la bitácora *listaCerrada* no necesita ser verificada y esta compuesta únicamente por nodos padres. Estos nodos son detallados más adelante. Los nodos específicamente a almacenar en cada una de las bitácoras se describen a continuación:

- En la bitácora *listaAbierta* se almacena el nodo de partida y sus nodos adyacentes. Si es la primera iteración, el nodo de partida es igual a q_{inicio}. Los nodos son almacenados siempre y cuando no sean parte de un obstáculo. Cada nodo agregado a la lista debe ser ligado a su nodo de partida mediante una etiqueta u otro recurso. Como objeto de esta relación, el nodo de partida es llamado nodo padre. No todos los nodos de partida son nodos padres, pero son candidatos a serlo. Al final, solo los nodos padres serán los que conformen la ruta buscada.

1	1	1
1	2	1
1	1	1

Figura 4.15: Matriz de conexión representación discreta. El número 2 representa la ubicación del robot y 1 los movimientos permitidos

- En la bitácora *listaCerrada* se almacenan los nodos padres, siempre y cuando se haya eliminado previamente de la bitácora *listaAbierta*.

Los dos criterios mencionados son estrictamente secuenciales. Para evidenciarlo, la Figura 4.16 describe gráficamente éste proceso. El cuadro verde en el centro es el nodo de partida, y está resaltada de color celeste para indicar que ha sido eliminada de la bitácora *listaAbierta* y agregada a la bitácora *listaCerrada*. Los cuadros negros de bordes verdes son los nodos agregados a la bitácora *listaAbierta*. Cada puntero blanco de los nodos indican su origen, es decir, el camino de regreso a su nodo padre.

Hasta este punto, se ha descretizado el entorno de búsqueda y construido bitácoras de los nodos. Ahora, como paso siguiente, es la ejecución de los dos pre-requisitos restantes. Cuando el etiquetado de la primera matriz de conexión ha culminado, una segunda es determinada. Para esto se selecciona uno de los

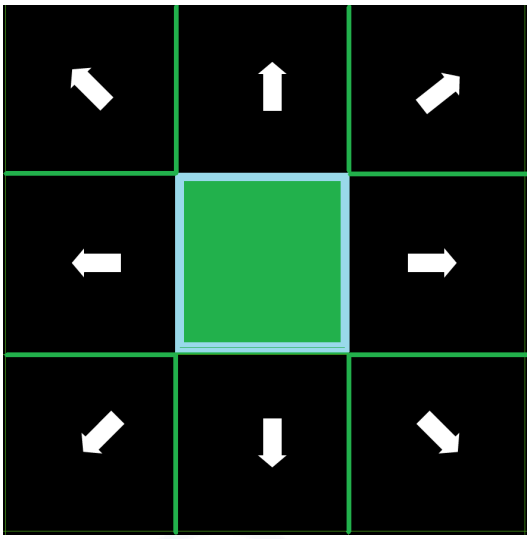


Figura 4.16: Clasificación de los nodos en bitácoras

nodos adyacentes de la *listaAbierta* como el nuevo nodo de partida. Aquel con el menor costo $F = G + H$.

El cálculo de F tiene implícito los dos pre-requisitos faltantes. Como se manifestó anteriormente, G es el costo requerido para trasladarse desde q_{inicio} a un nodo n dado. El costo G es de 10 para el movimiento vertical y horizontal, y 14 para el movimiento en diagonal. Asimismo, la heurística H, para este ejemplo, ha sido calculada estimando la distancia de Manhattan entre el nodo n y el nodo q_{meta}, con movimientos únicamente verticales y horizontales e ignorando los obstáculos que estén de por medio. Por lo tanto, el costo H de cada movimiento es de 10. Bajo este contexto, la Figura 4.17 ilustra los costos calculados para cada uno de los nodos adyacentes.

Una vez calculado los costos F de los nodos adyacentes, prosigue la selección del nodo de menor costo como el nuevo nodo de partida. De la Figura 4.17, el costo más pequeño de F es igual a 40. Sin embargo, el nodo con dicho costo es un caso particular de generación de nodos padres que vale la pena revisar. La Figura 4.18a muestra el nuevo nodo padre generado. Gráficamente, ya que los obstáculos y nodos padres se omiten, el nuevo nodo padre posee solo 4 nodos adyacentes y además los 4 nodos son compartidos con su homólogo. No obstante, teóricamente esto no es posible. Lo que se hace en estos casos es comparar costos. Si el nuevo nodo padre produce un costo menor al previamente calculado, el costo anterior es reemplazado y el nodo adyacente es asignado a su nuevo nodo padre. Por el contrario, si el nuevo nodo padre no produce costos más pequeños, como el ejemplo, éste es desechado y se busca un nuevo nodo padre. La Figura 4.18b

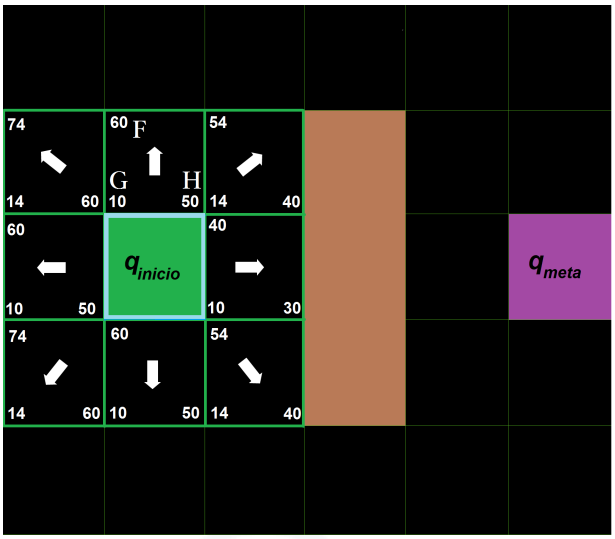
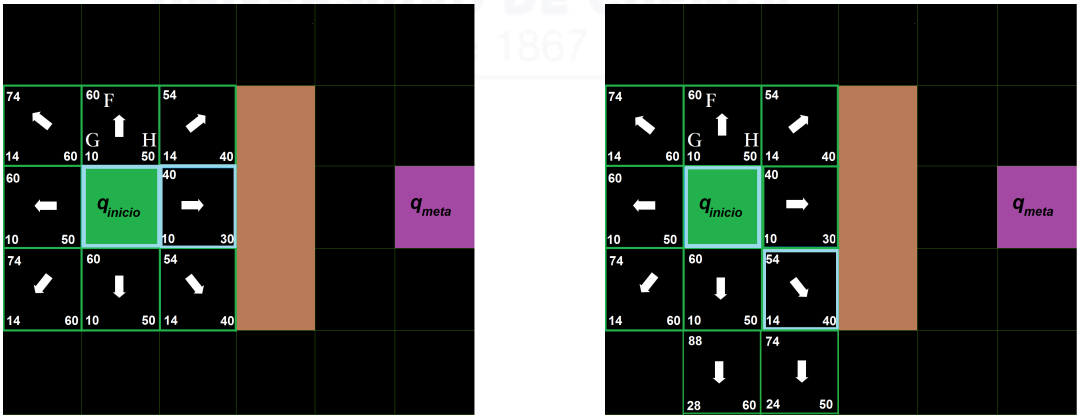


Figura 4.17: Cálculo de el costo total F

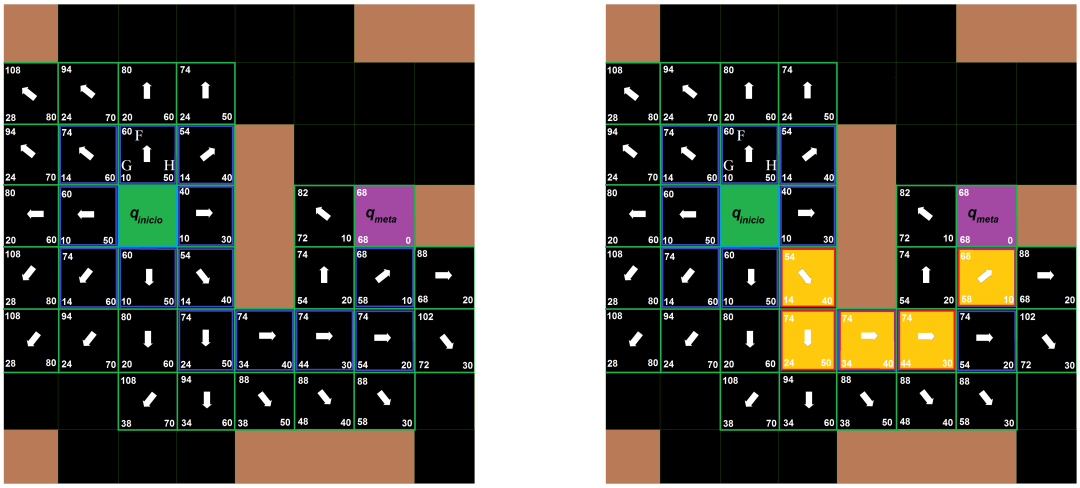
muestra el nodo padre desechado y el nuevo nodo padre generado, conjuntamente con las representaciones gráficas de las dos bitácoras.



(a) (b)

Figura 4.18: Generación de nodos padres

El proceso es repetido hasta que q_{meta} sea agregado a la bitácora $listaCerrada$, en dicho punto el resultado final de la búsqueda será semejante al escenario de la Figura 4.19a y la ruta más corta similar a los nodos de color amarillo de la Figura 4.19b.



(a) (b)

Figura 4.19: Resultados finales de la búsqueda de ruta más corta

Todos los procedimientos antes mencionados se engloban en el Código 4.4. El código resalta las partes básicas, pero necesarias, para el correcto funcionamiento del algoritmo. Asimismo, las simulaciones del algoritmo son mostradas en la sección subsiguiente, con ligeros cambios tanto de la heurísticas como de la matriz de conexión usada, pero que producen interesantes resultados.

Algoritmo 4.4: Pseudocódigo Algoritmo A Star

```
qinicio;
qmeta;
imagen;
% Parametro usado para manipular la influencia de H
D=10;

matrizConexion=[1 1 1;
                1 2 1;
                1 1 1];
qpartida=qinicio;

%listaAbierta=[qpartida G H F etiquetaPadre]
listaAbierta= [qincio 0 H(D,qinicio, qmeta) 0+H(D,qinicio, ...
    qmeta) =1];

listaCerrada[];
```

```
matrizAuxiliar=ones(size(imagen))

salir=true;

while salir

     nodoPartidaNuevo=obtenerMinimo(listaAbierta)
     borrarNodo(nodoPartida,listaAbierta);
     agregarNodo(nodoPartidaNuevo,listaCerrada,);
     if nodoPartidaNuevo=qmeta
        salir=false;
        break;
     [robotx, roboty]=find(matrizConexion==2);
     [nodosAdyaX, nodosAdyaY]=find(matrizConexion==1);

      for i=1:size(nodosAdyaX,1)
          %Nodo iterado
          nodoActual= [nodoPartidaNuevo(1)+ nodosAdyaX(i)-robotx
          nodoPartidaNuevo(2)+ nodosAdyaY(i)-roboty];

          if verificarCoision(nodoActual, nodoPartidaNuevo, ...
              imagen)
            if matrizAuxiliar(NodoActual(1),NodoActual(2))≠0

               Gactual=nodoPartida(3)+ G(nodoPartida(1:3),    ...
                  nodoActual);

               Hactual= heuristica(D, NodoAtual, qmeta);

               Factual= Gactual+Hactual

               listaAbierta= [listaAbierta; NodoActual Gactual ...
                  Hactual  size(listaCerrada,1)+1];

            end

          end

      end
      matrizAuxiliar(nodoPartidaNuevo(1),nodoPartidaNuevo(2))==0;
      %lista de nodos padres
      listaCerrada=[listaCerrada; nodoPartidaNuevo];
end
```

4.4.2. Simulaciones

El escenario a usarse para la evaluación del algoritmo A^* es el mostrado en la Figura 4.20. Para cumplir cabalmente el análisis del algoritmo, se ha propuesto combinaciones pares derivadas del tipo de matriz de conexión y heurística usada.

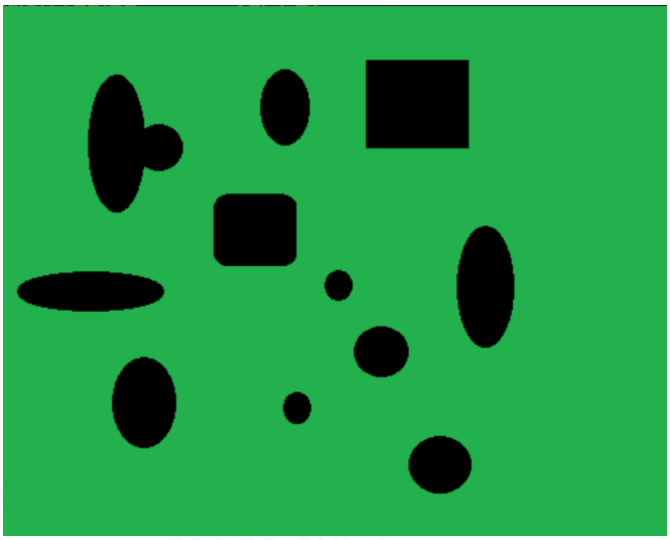
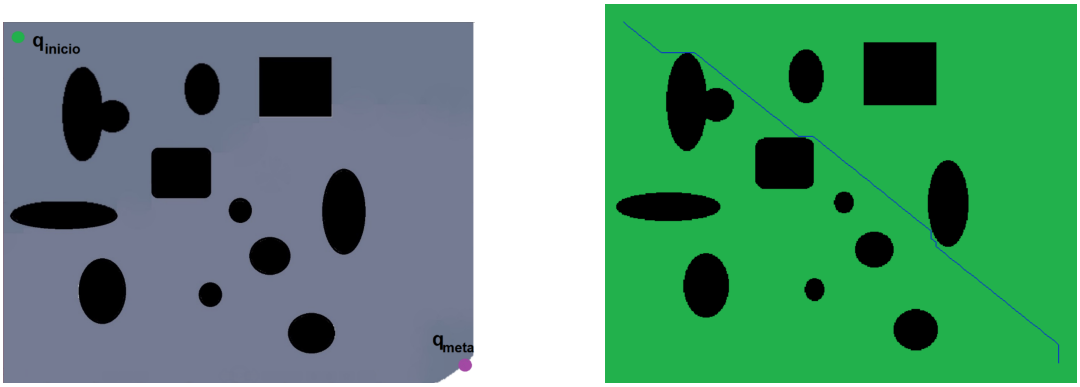


Figura 4.20: Escenario de simulación empleado para el algoritmo A^*

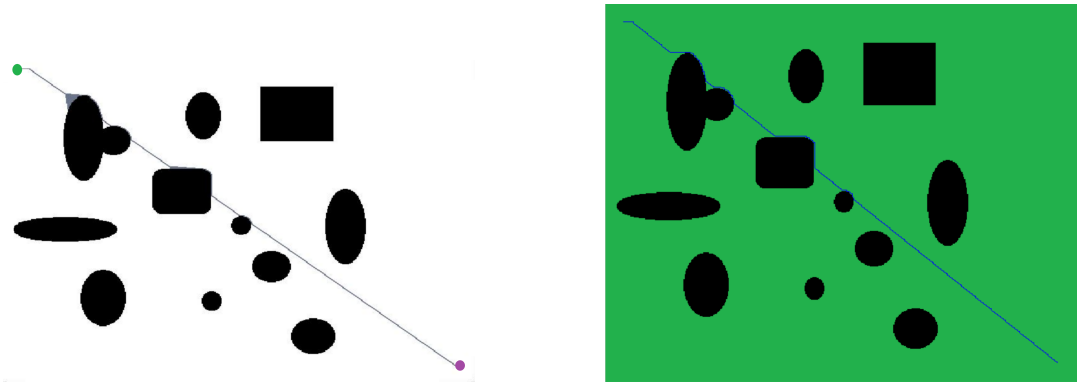
Para la primera combinación, se ha seleccionado la matriz de conexión de 8 grados de libertad y la heurística basada en la distancia de Manhattan, es decir, el usado en la explicación teórica. Inicialmente el robot se ubica en q_{inicio} con el reto de transitar hasta q_{meta}. El resultado de la travesía se ilustra en la Figura 4.21a. El color gris representa el área de búsqueda barrida por el algoritmo. Por lo tanto, se infiere directamente que la cantidad de nodos almacenados en la bitácora *listaAbierta* es considerablemente extensa, ocasionando un uso masivo de memoria y tiempos de procesamiento relativamente altos. El principal responsable éste comportamiento es la heurística de Manhattan. Dicha heurística es inadmisible y siempre estará segada a producir resultados similares a los ilustrados. Al final, la trayectoria descrita por los nodos padres se muestra en la Figura 4.21b, evidentemente muy acertada pero muy irregular y colindante con los obstáculos. Como veremos más adelante, estas dos propiedades no son consecuencia de la heurística usada, más bien es una propiedad inherente del algoritmo A^*.



(a) (b)

Figura 4.21: Resultado del algoritmo A^* con matriz de conexión de 8 grados de libertad y la heurística de Manhattan

Para la segunda combinación, la dupla seleccionada es una matriz de conexión de 4 grados de libertad conjuntamente con la heurística Distancia Diagonal. De la misma manera, el robot es ubicado en el mismo q_{inicio} y con el mismo destino q_{meta}. La Figura 4.22a es el barrido generado por el algoritmo y la Figura 4.22b la ruta final encontrada. Claramente los nodos indexados a la bitácora *listaAbierta* son casi únicamente los usados por la matriz de conexión, ésto congrega un incremento radical de la velocidad de convergencia y una disminución considerable de recursos. Todo gracias a la admisibilidad de la heuristica Distancia Diagonal. Como se menciono anteriormente, no importa la heurística usada, aun persiste el problema de la proximidad de obstáculos e irregularidad de la ruta.

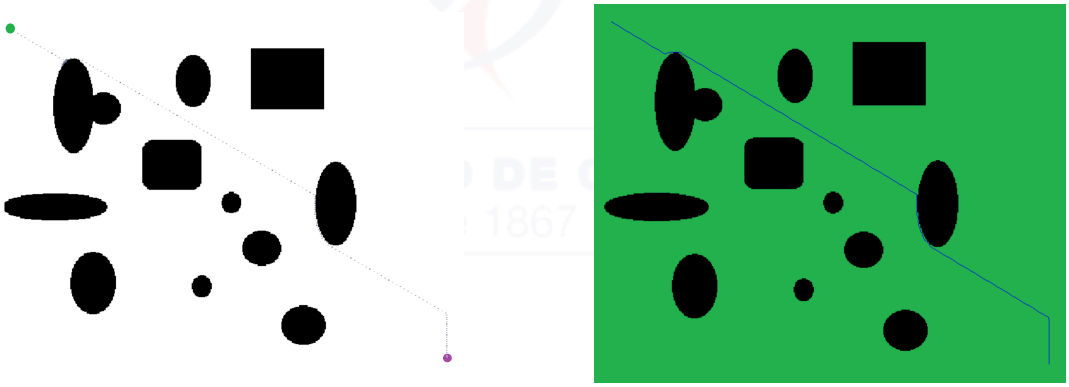


(a) (b)

Figura 4.22: Resultado del algoritmo A^* con matriz de conexión de 4 grados de libertad y la heurística Distancia diagonal

Finalmente, bajo las mismas condiciones, la Figura 4.23 muestra los resulta-

dos de simular el par matriz de conexión sin restricción y la heurística Distancia Euclideana. La similitud de resultados con la combinación anterior es mucha, sin embargo, ésta almacena mucho menos nodos en la bitácora *listaAbierta*, a tal punto que la ruta de gris explorada, Figura 4.23a, es casi imperceptible. En consecuencia, ésta combinación resulta ser la más rápida de todas. Para finalizar, cabe mencionar que las combinaciones sustentadas en éste trabajo de titulación no fueron tomadas al azar, más bien escogidas de entre un grán número de simulaciones realizadas previamente. De las misma manera, la heurística Distancia Diagonal produce los mismos resultado sin importar la matriz de conexión, pero entre más grados de libertad posea la matriz es mejor. La heurística de Distancia Euclidiana solo provee buenos resultados si trabaja conjuntamente con matrices de conexión sin restricciones.



(a) (b)

Figura 4.23: Resultado del algoritmo A^* con matriz de conexión sin restricción y la Distancia Euclidiana

En conclusión, el algorimto A^* usa heurísticas para controlar su comportamiento, objetivando la búsqueda rápida y óptima. Desafortunadamente, el algoritmo solo garantiza una solución óptima si su función heurística es admisible. Por otra parte, el algoritmo genera rutas fragosas y muy cercanas a obstáculos, ocasionando un serio riesgo al tránsito del robot DaNI. Bajo este contexto, el siguiente apartado presenta una técnica basada en la propagación de interfaces que anula dichos inconvenientes a costa de un ligero incremento del procesamiento.

4.5. Método fast marching

La propagación de interfaces es una técnica que ha revolucionado el campo de la ingeniería, teniendo numerosos campos de aplicación: fluidos, combustión, manufactura de semiconductores, sismología, visión por computadora, segmentación de imágenes, navegación de robots, planificación de de rutas, etc. Éste trabajo de titulación lo emplea en el área de la planificación de rutas.

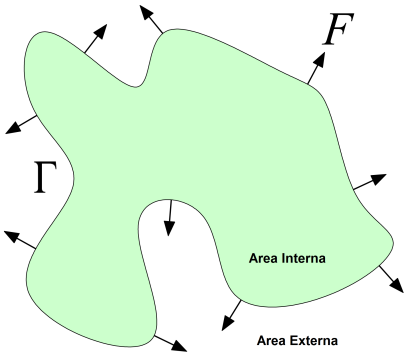


Figura 4.24: Frente en dos dimensiones moviendose a una velocidad F [15].

Con el objeto de profundizar el estudio de la propagación de interfaces, es necesario aclarar el concepto de interfaz y los términos relacionados a ella. Dada una región arbitraria, Figura 4.24, la interfaz se define como la curva, o contorno, que separa el área interna del área externa. La interfaz puede propagarse de dos formas: expandirse o contraerse estrictamente, y expandirse y contraerse arbitrariamente, dependiendo de la naturaleza de la velocidad. Vectorialmente hablando, existen tres componentes de velocidad definidas: una componente local parte de la interfaz, una componente relacionado a las propiedades globales de la superficie y una componente independiente de la superficie. No obstante, la mayoria de bibliografías recomiendan ignorar las direcciones separadas de estas velocidades y simplemente usar una función escalar F (función de velocidad) para describir la velocidad normal a la interfaz Γ, Figura 4.24. La propagación de interfaces está gobernada por dos criterios: formulación de falor inicial y formulacion de valor de contorno [62–64].

Formulación de valor inicial

La evolución de una interfaz o frente que aleatoriamente se expande y contrae, es decir con velocidad F arbitraria, puede ser estudiado resolviendo la ecuación

diferencial parcial de valor inicial. Dicha ecuación se obtiene de la evolución de los conjuntos de nivel de una funcion. El conjunto de nivel de una función $f(\mathbf{x})$, $\mathbf{x} \epsilon \mathbf{R}^n$, es definido como el conjunto de puntos en los cuales la función toma un valor constante especifíco; matemáticamente, el conjunto de nivel de $f(\mathbf{x})$ es el conjunto, $\{\mathbf{x}|f(\mathbf{x}) = C\}$, donde C es una constante dada. La idea original detrás de los métodos conjunto de nivel es muy simple. Dado un frente Γ en $\mathbf{R}^{n-1}$, una función Conjunto de Nivel, $\Phi(\mathbf{x}, t)$, es definida en $\mathbf{R}^n$ tal que el frente Γ forme el conjunto de nivel cero de la función $\Phi(\mathbf{x}, t)$ [62]. De esta manera, cada valor de t genera la expansión de Γ, y la familia de interfaces queda totalmente representada por $\Gamma(t)$. La Figura 4.25 ilustra el empotramiento de un frente en dos dimensiones como el conjunto de nivel cero de un campo escalar de tres dimensiones $\Phi(\mathbf{x}, t)$.

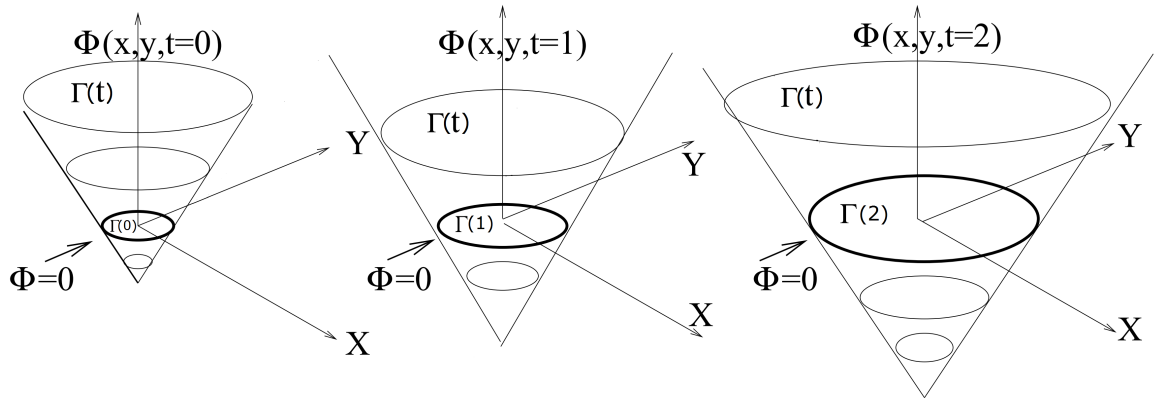


Figura 4.25: Problema de valor inicial [16]

La Ecuación 4.2 describe el empotramiento de un frente.

$$\Phi\left(\mathbf{x}\left(t\right), t\right) = 0 \tag{4.2}$$

A su vez, la evolución de Φ, aplicando la regla de la cadena a la ecuación anterior, está modelado por la Ecuación 4.3.

$$\Phi_t + \nabla\Phi\left(x\left(t\right), t\right) x^{'}\left(t\right) = 0 \tag{4.3}$$

donde Φ_t representa la primera derivada de Φ. Reescribiendo la ecuación, ya que $\mathbf{F}$ es la velocidad normal a Γ, donde $x^{'}\left(t\right)\mathbf{n} = \mathbf{F}$ con $\mathbf{n} = \frac{\nabla\Phi}{|\nabla\Phi|}$ resulta,

$$\Phi_t + \mathbf{F}\left|\nabla\Phi\right| = 0 \tag{4.4}$$

$$\Phi\left(x, t = 0\right) = C \tag{4.5}$$

Finalmente, la formulación de valor inicial completa se resume en las siguientes condiciones,

$$\Phi_t + \mathbf{F}\,|\nabla\Phi| = 0$$
$$Frente = \Gamma\,(t) = \{(x,y)\,|\Phi\,(x,y,z) = 0\}$$
$$Aplicado\ a\ \mathbf{F}\ arbitrarias$$

La Ecuación 4.4 es típicamente resuelta usando métodos numéricos sobre un dominio discretizado. No obstante, el inconveniente de los métodos conjunto de nivel radica en su gasto computacional. Ya que el frente es empotrado en una dimension mayor, esto introduce un esfuerzo computacional adicional. Afortunadamente, existe el problema de valor de frontera, caso particular del problema de valor inicial, que reduce significativamente la resolución de la ecuación diferencial parcial de valor inicial.

Formulación de valor de frontera

El problema de valor inicial puede ser convertido al problema de valor de frontera. Para esto, se asume que la interfaz es monótona creciente (o decreciente) [65], en otras palabras, que la velocidad F solo puede tomar valores positivos (o negativos). Si esta condición se cumple siempre, un nuevo parámetro $T\,(x,y)$, llamado tiempo de arribo, puede ser introducido,

$$\Phi\,(x,y,T\,(x,y)) = 0 \tag{4.6}$$

La ecuación 4.6 significa que el frente del conjunto de nivel cero óptimo alcanza el punto (x,y) en el tiempo $T\,(x,y)$, como se ilustra en la Figura 4.26. Por lo tanto,

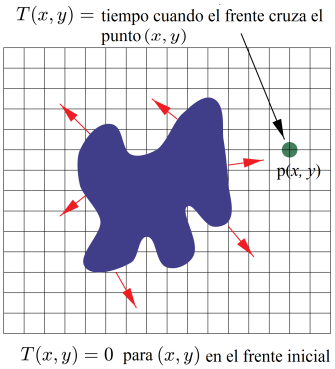


Figura 4.26: Tiempo de arribo T [17]

la meta es construir una superficie solución $T(x, y)$ fuera del valor de la curva frontera, donde cada conjunto de nivel de T determine el frente Γ_T. El proceso se ilustra en la Figura 4.27. La formulación de valor de frontera cuenta con dos versiones. Si la función de velocidad F solo depende de la posición y las primeras derivadas de la solución T, resulta una ecuación estática Hamilton-Jacobi [66]. Por el contrario, si la función velocidad F solo depende de la posición (x, y),

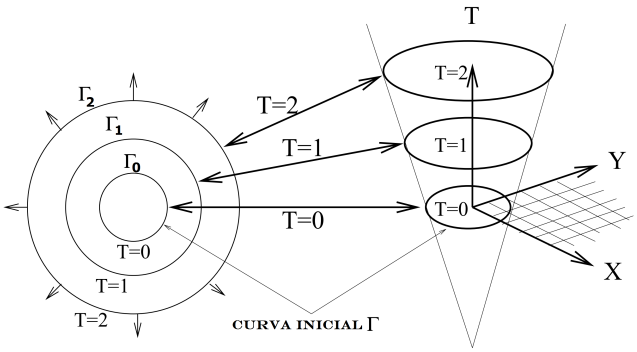


Figura 4.27: Problema de valor de frontera [16]

resulta la ecuación estacionaria Eikonal. Para nuestro propósito, solamente nos enfocaremos en la ecuación Eikonal, Ecuación 4.7.

$$F\left|\nabla T\left(x, y\right)\right| = 1 \tag{4.7}$$

Mientras la velocidad sea $F > 0$ o $F < 0$ y ésta dependa únicamente de la posición (x, y), la ecuación Eikonal estacionaria puede ser obtenida de la Ecuación 4.4. La demostración matemática supera el alcance de este documento. Para mayor información, Sri Venkata detalla toda la deducción matematica inherente al proceso en [62]. En conclusión, las condiciones del problema de valor de frontera se resumen en,

$$F\left|\nabla T\left(x, y\right)\right| = 1$$
$$Frente = \Gamma\left(t\right) = \left\{\left(x, y\right)|T\left(x, y\right) = t\right\}$$
$$Requiere\ F > 0$$

La ventaja de esta pespectiva, es que la ecuación Eikonal se puede resolver numéricamente utilizando FMM [63, 67, 68]. El método FMM, introducido en [65], posee virtudes tales como trabajar en espacios de dimensiones arbitrarias, computo preciso y eficiente del movimiento de frentes. FMM aproxima la solución de

ecuaciones diferenciales parciales de valor de frontera, en base a la interfaz de propagación.

Método fast marching

Sethian propuso una solución discreta para resolver la ecuacion Eikonal, conocido como el Método Fast Marching [65]. En el caso bidimensional, el entorno es discretizado usando un mapa de ocupación, correspondiente al conjunto de puntos $p(x_i, y_j)$; donde i, j son la fila y columna de cada nodo (píxel o celda) respectivamente.

$$|\nabla T(x,y)| = \frac{1}{F(x,y)}$$

La discretización propuesta del gradiente $|\nabla T(x,y)|$, se muestra en la Ecuación 4.8.

$$max\left(D_{ij}^{-x}T, 0\right)^2 + min\left(D_{ij}^{+x}T, 0\right)^2 + max\left(D_{ij}^{-y}T, 0\right)^2 + min\left(D_{ij}^{+y}T, 0\right)^2 = \frac{1}{F_{ij}^2} \quad (4.8)$$

A su vez, de la ecuación anterior, se puede obtener una version más sencilla, a costa de una precisión menor,

$$max\left(D_{ij}^{-x}T, -D_{ij}^{+x}, 0\right)^2 + max\left(D_{ij}^{-y}T, -D_{ij}^{+y}, 0\right)^2 = \frac{1}{F_{ij}^2} \quad (4.9)$$

donde:

$$D_{ij}^{-x} = \frac{T_{i,j} - T_{i-1,j}}{\Delta x}$$
$$D_{ij}^{+x} = \frac{T_{i+1,j} - T_{i,j}}{\Delta x}$$
$$D_{ij}^{-y} = \frac{T_{i,j} - T_{i,j-1}}{\Delta y}$$
$$D_{ij}^{+y} = \frac{T_{i,j+1} - T_{i,j}}{\Delta y}$$

con $T_{i,j}$ como el tiempo de llegada a la celda (i, j), y Δx y Δy como el incremento de espacio en las direcciones x e y. Si los parámetros D son sustituidos en la

Ecuación 4.9, se obtiene,

$$\begin{aligned} T &= T_{i,j} \\ T_1 &= min\left(T_{i-1,j}, T_{i+1,j}\right) \\ T_2 &= min\left(T_{i,j-1}, T_{i,j+1}\right) \end{aligned}$$

Entonces, la ecuación Eikonal para un espacio bidimensional, en su forma discreta, queda definida por la Ecuación 4.10.

$$max\left(\frac{T-T_1}{\Delta x}, 0\right)^2 + max\left(\frac{T-T_2}{\Delta y}, 0\right)^2 = \frac{1}{F_{ij}^2} \tag{4.10}$$

Consecuentemente, ya que para nuestro caso se asume que la velocidad toma valores positivos, $F > 0$, T debe ser mayor que T_1 y T_2 siempre y cuando la onda no haya pasado previamente por las coordenadas i, j. Así, eventualmente, la Ecuación 4.10 se reduce a la resolución de una ecuación cuadrática, Ecuación 4.11.

$$\left(\frac{T-T_1}{\Delta x}\right)^2 + \left(\frac{T-T_2}{\Delta y}\right)^2 = \frac{1}{F_{ij}^2} \tag{4.11}$$

Como la onda se propaga desde varios puntos hacia otro en particular, siempre se mantendrá el T mínimo obtenido al resolver la ecuación cuadrática. Por el contrario, si resulta que $T < T_1$ o $T < T_2$, en concordancia con la Ecuación 4.10, se sustituirá su valor por cero. Entonces, si $T < T_1$,

$$\left(\frac{T-T_2}{\Delta y}\right) = \frac{1}{F_{ij}} \tag{4.12}$$

Caso contrario, si $T < T_2$, entonces

$$\left(\frac{T-T_1}{\Delta x}\right) = \frac{1}{F_{ij}} \tag{4.13}$$

Finalmente, la ecuación Eikonal discreta se resuelve de forma iterativa para encontrar el mapa solución T, que contendrá el tiempo de llegada a cada píxel relativo al origen de la interfaz. Si se considera el valor de tiempo de llegada, T, como una tercera dimensión, la superficie solución T sería similar al de la Figura 4.28, donde cada interfaz esta formada por todos los puntos que han pasado durante el mismo instante de tiempo (Isocurvas).

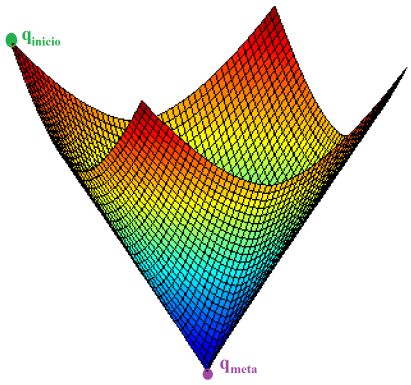


Figura 4.28: Superficie $T(x, y)$ [13]

Gradiente de máxima dirección

Una vez encontrado el mapa solución T, el paso siguiente es determinar la ruta óptima . Dada las características de FMM, la interfaz generada proporcionará el camino óptimo entre el punto de inicio y la meta en términos de tiempo . Además, si la velocidad es constante, la ruta será también óptima en cuanto a distancia. El criterio usado para calcular la ruta óptima es el gradiente de máxima dirección. Ya que la interfaz se origina desde la meta, $T = 0$, el mapa solución T tendrá un mínimo en dicho punto. Este hecho es más notorio en la superficie T de la Figura 4.28. Por lo tanto, siguiendo la dirección del gradiente máximo desde el punto de inicio a la meta, se determina la ruta óptima.

El gradiente de máxima dirección está ponderada como el valor medio de sus dos vecinos, tanto para x y y. Para dos dimensiones, el cálculo sería,

$$grad_x = \frac{T_{i-1,j} + T_{i+1,j}}{2} \tag{4.14}$$

$$grad_y = \frac{T_{i,j-1} + T_{i,j+1}}{2} \tag{4.15}$$

La ruta es calculada iterativamente, calculando los valores de $grad_{xi}$ y $grad_{yj}$ para cada punto $p_i\,(p_{xi}, p_{yi})$. Por lo tanto, usando p_i se calcula p_{i+1}. Es decir,

$$p_{x(i+1)} = p_{xi} + mod_i * cos\,(alpha_i) \tag{4.16}$$

$$p_{y(i+1)} = p_{yi} + mod_i * cos\,(alpha_i) \tag{4.17}$$

donde:

$$mod_i = \sqrt{(grad_{xi})^2 + (grad_{yi})^2}$$
$$alpha_i = \left(\frac{grad_{yi}}{grad_{xi}}\right)$$

Con todos los criterios antes mencionados, el algoritmo FMM cumple con los requisitos necesarios para su aplicación en el campo de la planificación de rutas, otorgando eficiencia y rutas llanas.

4.5.1. Funcionamiento

Similar al algoritmo A^*, el proceso iterativo FMM emplea etiquetas para cada una de los nodos con el fin de identificar su estado actual. Existen 3 posibles estados:

- Nodos desconocidos: Representan las celdas cuyo valor T no esta definido.
- Nodos banda angosta: Son las celdas que ya tienen asignados un valor T, pero puede variar, similar a la $listaAbierta$ en el algoritmo A^*. Por estas celdas se expandirá la onda en las siguientes iteraciones. Estos nodos representa la interfaz.
- Nodos congelados: Son las celdas con un valor fijo de tiempo T, por las cuales la onda ha pasado ya, similar a la $listaCerrada$ en el algoritmo A^*.

A su vez, el desarrollo del algoritmo se dividen en tres pasos:

- **Inicialización**: Todos los puntos fuentes de onda son etiquetados como congelados y asignados el valor $T = 0$. Para simplificar la explicación, solo se considera un punto fuente, siendo la meta. Seguidamente, los vecinos al punto fuente son etiquetados como nodos banda angosta y se calcula su tiempo de llegada. Pero para esto, el mapa de velocidad F es requerido. Ventajosamente, ya que la velocidad F debe ser mayor a cero y el entorno puede ser representado como un mapa binario con valores 1 y cero (celdas libres y obstaculos), la velocidad puede tomar directamente estos valores. Por lo tanto, la onda tendrá una velocidad constante de 1 en las celdas libres, mientras que en las celdas con obstáculos la onda nunca se propagará, ya que el tiempo de llegada será infinito según la Ecuación 4.10.

Algoritmo 4.5: Pseudocódigo Algoritmo FMM [13]

```
Entrada: Imagen G  % mxn
Entrada: Mapa de Velocidad F  % mxn
Entrada: Conjunto Orig de Puntos fuentes % Numero de ondas
Salida:  Mapa Gij de Tiempo T  % tiempos de llegada a cada celda
%INICIALIZACION
for Gij pertenece Orig
    Gij.T <-- 0;
    Gij.estado <-- CONGELADO;
    for Gkl pertenece Gij.Vecinos
       if Gkl ≠ CONGELADO
          Gkl.T <-- ResolverEcuacionEikonal(Gkl);
          if Gkl.estado = BANDAANGOSTA
             Banda_Angosta.ActualizarPosicion(G_kl);
          end
          if Gkl.estado = DESCONOCIDO;
              Gkl.estado <-- BANDAANGOSTA
              Banda_Angosta.InsertarEnPosicion(Gkl);
          end
       end
    end
%BUCLE
    while Banda_Angosta NoVacio
       Gij <-- Banda_Angosta.MejorPrimero();
       for Gkl pertenece Gij.Vecinos
          if Gkl ≠ CONGELADO
             Gkl.T <-- ResolverEcuacionEikonal(Gkl);
             if Gkl.estado = BANDAANGOSTA
                Banda_Angosta. ActualizarPosicion(G_kl);
              end
              if Gkl.estado = DESCONOCIDO
                Gkl.estado <-- BANDAANGOSTA;
                Banda_Angosta.InsertarEnPosicion(Gkl);
              end
          end
       end
    end
end
```

- **Bucle Principal**: De forma secuencial se resuelve la ecuación Eikonal discreta para cada uno de los nodos etiquetados como banda agosta, almacenando el menor valor de tiempo de llegada T. Una vez terminado el proceso, cada uno de los nodos son etiquetados como congelados.

- **Finalización**: Cuando todos los nodos han sido etiquetados como congelados, y ya no existan nodos banda angosta, el algoritmo concluye.

Graficamente, los tres pasos mencionados anteriormente se resumen en la Figura 4.29 y su pseudocódigo se muestra en el Algoritmo 4.5. El proceso de expansión a través de los nodos es tal cual una onda lo haría en un medio físico homogéneo. Por ejemplo en el agua. Sin embargo, si el medio no es homogéneo, las ondas se deformarían, debido a las diferentes componentes de velocidad existentes.

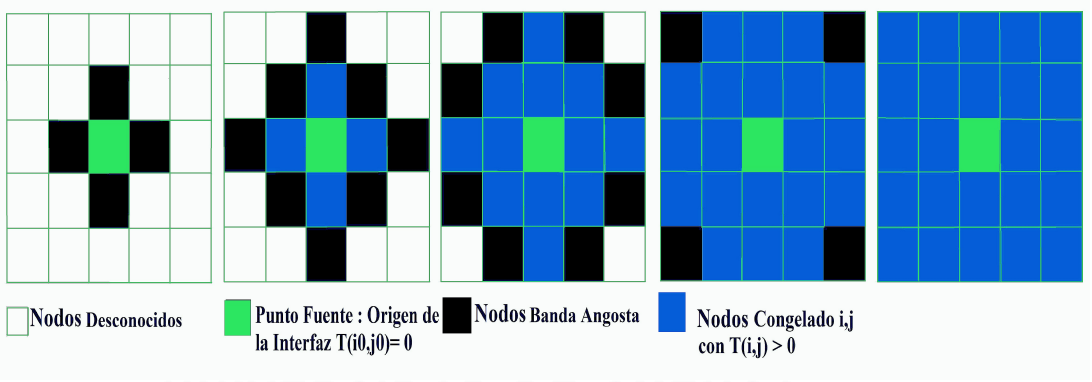
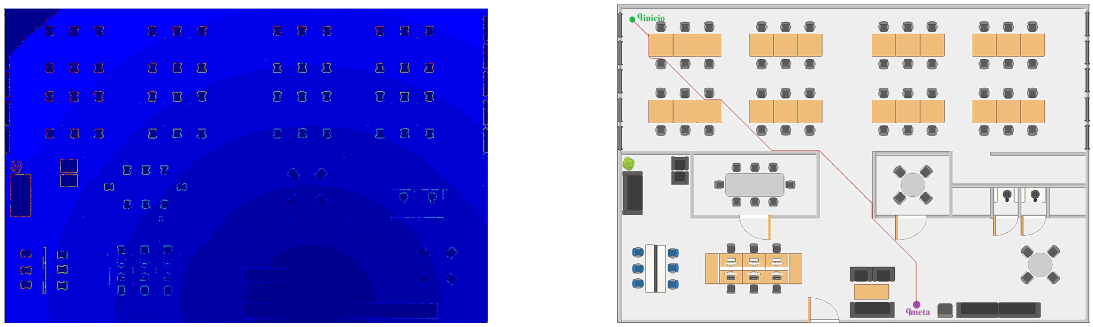


Figura 4.29: Descripción gráfica del algoritmo FMM en un mapa de $5x5$

4.5.2. Simulación

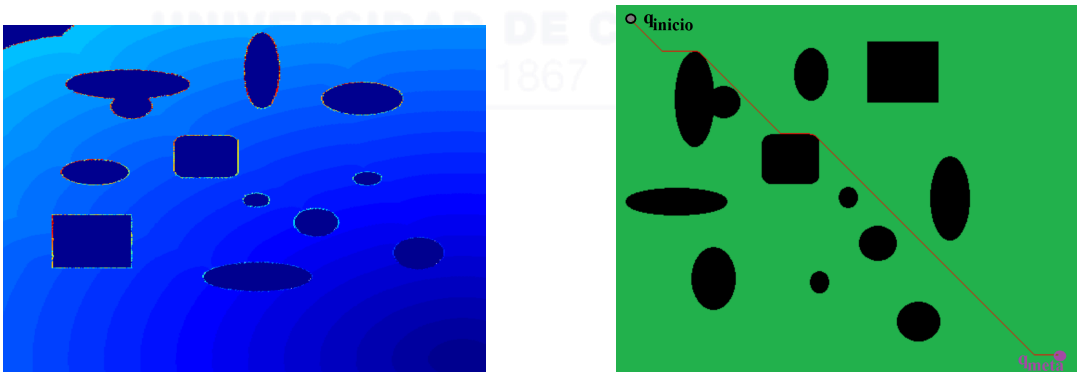
La intención de ésta sección es realizar una comparativa entre las rutas generadas por los algoritmos RRT, $B-RRT$, A^* y FMM. La principal diferencia entre el algoritmo FMM y sus homólogos, es la generación de rutas llanas. Si se considera el escenario 4.4a, el mismo usado en RRT y $B=RRT$. Al aplicar el algoritmo FMM, la ruta generada no tiende a contornear a los obstáculos, mas bien, intenta inmediatamente evadirlo, generando un conjunto de segmentos rectilíneos como ruta. La Figura 4.33b ilustra esta ventaja. A su vez, la Figura 4.33a muestra la generación de ondas, iniciando en la meta y finalizando en el punto de partida.



(a) (b)

Figura 4.30: Simulación del algoritmo FMM

Análogamente, el escenario 4.20, usado con el algoritmo A^*, presenta una ruta con similares características formando un conjunto de segmentos rectilíneos. La Figura 4.31a representa la expansión de la onda y la Figura 4.31b la ruta generada.



(a) (b)

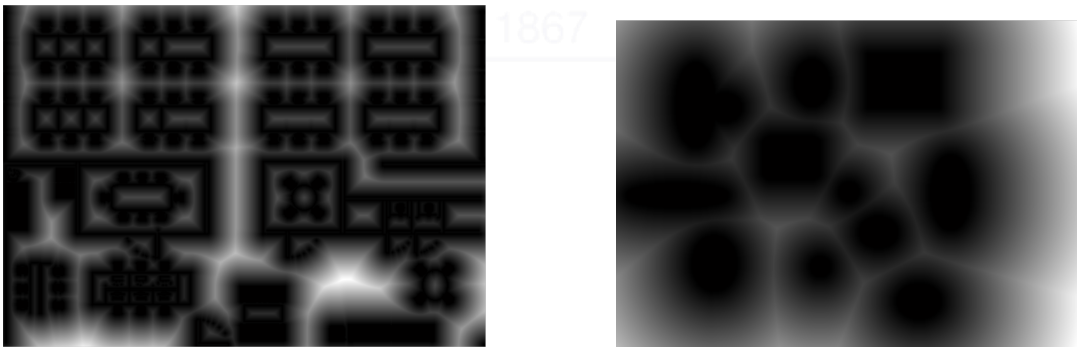
Figura 4.31: Simulación del Algoritmo FMM

En conclusión, los algoritmos RRT, $B-RRT$ y A^* ofrecen una gran velocidad de procesamiento a costa de rutas demasiado irregulares y colindantes con los obstáculos. En cambio, el algoritmo FMM incrementa procesamiento apremiando una ruta modesta. Pero, aunque la ruta generada muestra una circunstancial mejora, el espaciado entre la ruta y los obstáculos sigue siendo un problema. Con esto en mente, la siguiente sección analiza el algoritmo fast marching square, variante del algoritmo FMM, que suma la posibilidad de manipular el espaciado entre la ruta generada y los obstáculos; sin perder eficiencia y suavidad.

4.6. Fast marching square

La gran mayoría de los algoritmos dedicados a la planificación de trayectorias son propensos a generar rutas muy pegadas a los obstáculos. Éste inconveniente genera peligros para el robot DaNI, ocasionando continuos choques que eventualmente causaran algún daño. Además, los errores en la generacion de movimiento y medicion de los sensores, hacen vital una distancia de seguridad. Desafortundamente, los 4 algoritmos estudiados hasta aquí se adjudican tal problema, y es casi imposible usarlos como planificadores de trayectorias. Para solucionar este tipo de problemas se propone el algoritmo Fast Marching Square (FMM^2, por sus siglas en inglés).

FMM^2 es un algoritmo robusto y eficiente para el cálculo de trayectorias seguras y uniformes. La potencia de este algoritmo ha sido demostrada durante los últimos años aplicado a una gran variedad de problemas de planificación de trayectorias, tales como: planificación de formación autómata, aprendizaje cinemático, generación de mapas de rutas, etc.



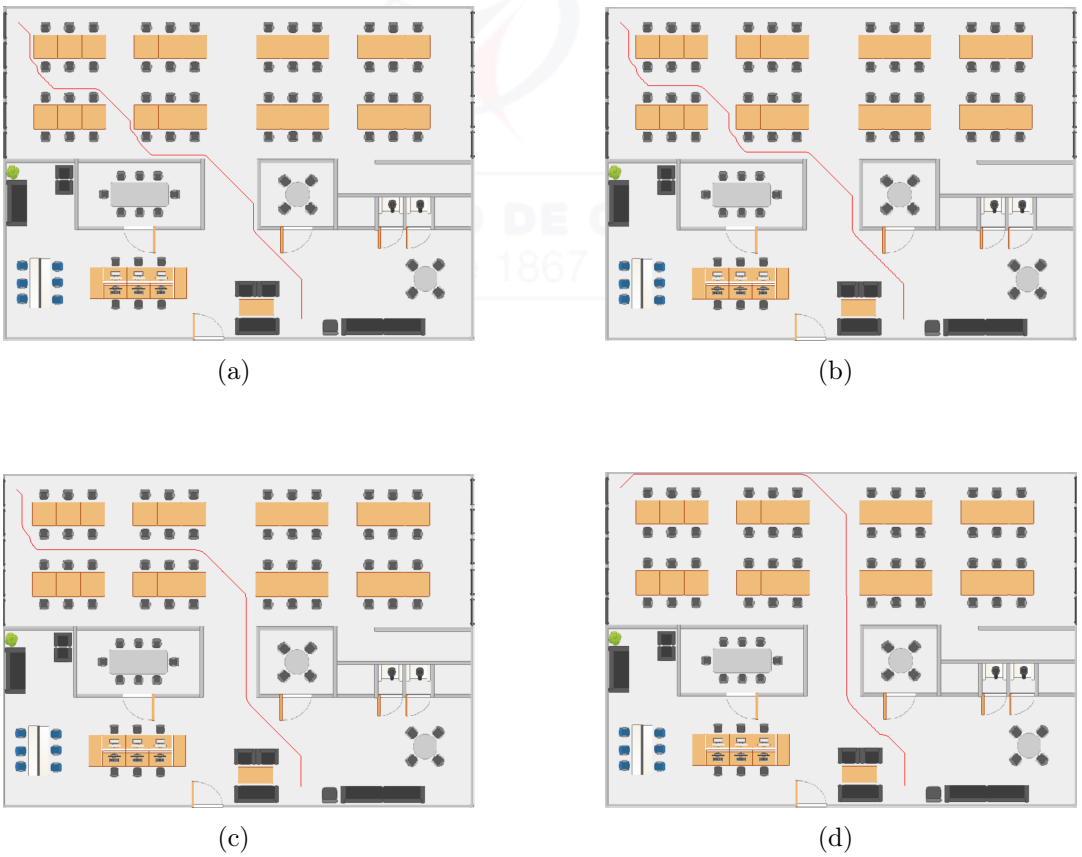
(a) (b)

Figura 4.32: Mapas de velocidad en función de la posición

El principio de FMM^2 cosiste en generar un mapa de velocidad, teniendo en cuenta la cercanía de los obstáculos para modificar la expansión de la interfaz. Por tanto, ya no se asume la velocidad como un mapa binario, como suele hacerlo el algoritmo FMM. Ahora, para generar el mapa de velocidades, FMM^2 propone utilizar cada obstáculo como fuente de una onda, generando una gran multitud de frentes al mismo tiempo. Así, mientras las ondas se alejen de los obstáculos, el tiempo T se incrementa. Esta propiedad se puede interpretar como el mapa de velocidad F de la imagen. De esta manera, ya que las velocidades serán menores cuanto más cerca de los obstáculos y más grandes conforme se alejan, los tiempos

óptimos estarán ubicados en lugares donde la velocidad tenga valores grandes (lejos de los obstáculos); debido a su relación inversamente proporcional. Este concepto es sobre el cual se cimienta el algoritmo FMM^2, para esto, el algoritmo FMM debe ser ejecutado por duplicado para encontrar el mapa de velocidades F y el tiempo T respectivamente. La Figura 4.32 muestra el mapa de velocidades para los entornos de las Figuras 4.4a y 4.20.

Luego, la trayectoria se puede calcular mediante los pasos detallados en la sección anterior, con la excepción de la velocidad F que pasa de ser constante (1 binario) a variable, implicando inequívocamente que la velocidad de expansión depende de la posición. Al final, la expansión de la onda a velocidad variable proporcionará una trayectoria óptima en tiempo.



(a) (b)

(c) (d)

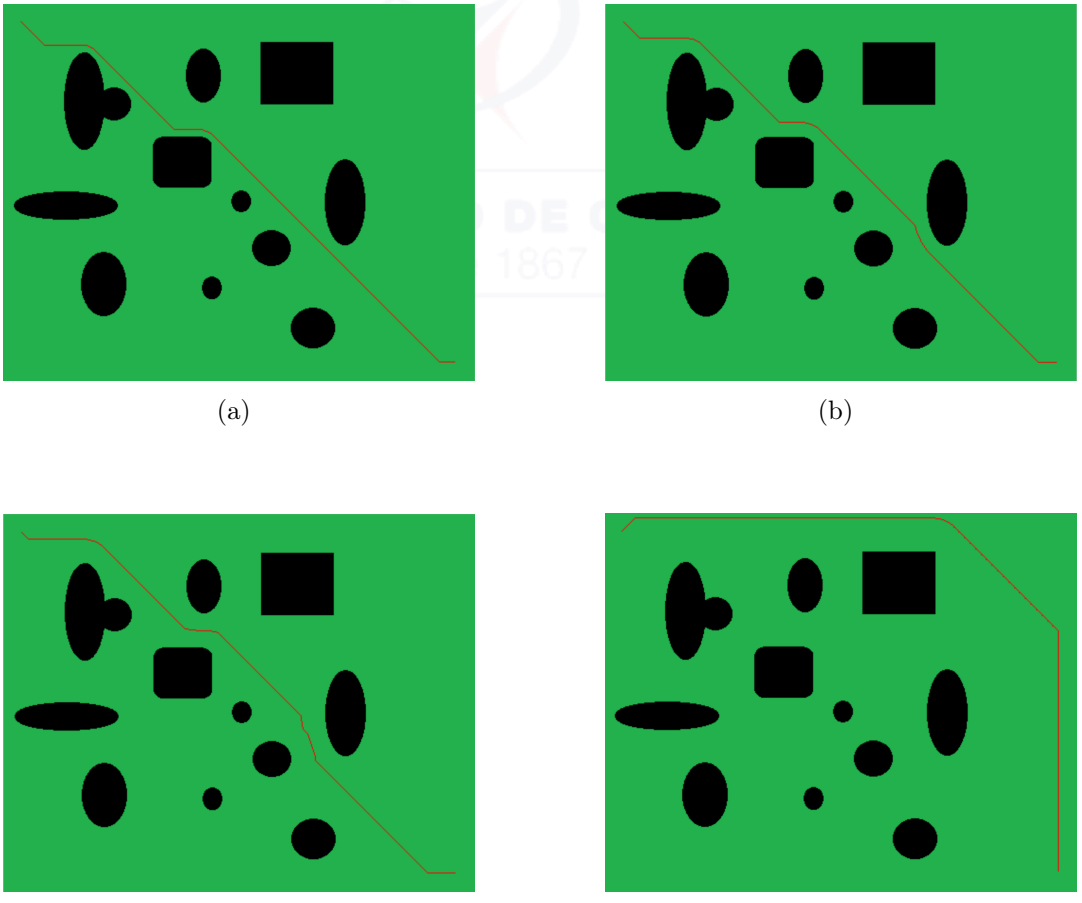
Figura 4.33: Aplicación del algoritmo FMM^2

4.6.1. Simulaciones

La Figura 4.33 muestra 4 rutas para el mismo escenario, punto de partida y meta. Para los 4 casos las rutas son óptimas en términos de tiempo, la dife-

rencia entre uno y otro es el espaciado existente entre la ruta y los obstáculos. Las Figuras 4.33a, 4.33b, 4.33c y 4.33d representan el incremento progresivo del espaciamiento que demandan 4 robots de diferente volumen. Aunque FMM^2 permite incrementar el espaciamiento a voluntad, hay que ser muy cuidadosos con su manipulación, caso contrario, la ruta se sobredimensionará. Por ejemplo, para nuestro caso, la ruta candidata para el transito del robot DaNI es el de la Figura 4.33c, así la ruta de la Figura 4.33d no sería óptima para el robot.

De igual manera, la Figura 4.34 muestra el incremento progresivo del espaciamiento para el entorno 4.20, constatando la invaluable propiedad que posee el algoritmo FMM^2 y que muy pocos lo poseen. Por tal motivo, el algoritmo es catalogado como una potencia en el campo de la planificación de rutas.



(a) (b)

(c) (d)

Figura 4.34: Aplicación del algoritmo FMM^2

4.7. Conclusión

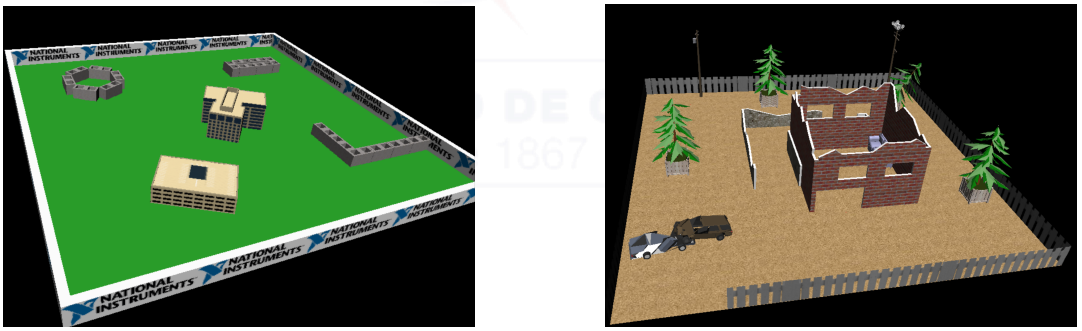
Este capítulo expone los fundamentos teóricos necesarios para el entendimiento e implementación de las diferentes alternativas que existen en el campo de la planificacion de rutas. No todos los métodos propuestos en éste trabajo de titulación satisfacen los criterios de como un robot debería navegar en un entorno. Las limitaciones de los algoritmos radican en que la ruta generada, a pesar de ser eficiente, generan rutas que comprometen la integridad del robot, siendo irregulares y poco suaves. Por ejemplo, los algoritmo RRT y $B-RRT$, a pesar de computacionalmente eficientes, generan rutas irregulares. El algoritmo A^* genera rutas regulares, pero el tiempo de procesamiento está estrechamente ligado al tipo de heurística usada (heurísticas admisible). Si se considera la heurística de Manhattan (heurística no admisible), sin importar la matriz de conexión usada, el tiempo de procesamiento se ve considerablemente comprometido. Como heurísticas próximas a una admisible se recomiendan la heuristica Distancia Diagonal y Distancia Euclideana. El algoritmo FMM genera rutas regulares pero de entre los 3 anteriores el costo computacional es mayor. Además, las rutas generadas por los algoritmos RRT, $B-RRT$, A^* y FMM están sesgadas a colindar con los obstáculos, no así el algoritmo FMM^2.

Por lo tanto, FMM^2 posee un gran potencial para resolver problemas que requieren cierto espaciamiento entre la ruta y los obstáculos, generando rutas óptimas en tiempo y espacio. Además, gracias a la velocidad obtenida en cada punto, es posible realizar un análisis cinemático del robot, determinando las velocidades adecuadas de transito del robot para cada tramo de la ruta.

Capítulo 5

Pruebas y Resultados

Las pruebas del UGV desarrollado sobre DaNI se realizaron de forma virtual mediante simulaciones y a través de experimentos en un ambiente cerrado controlado. Los resultados se presentan en las secciones siguientes.



(a) (b)

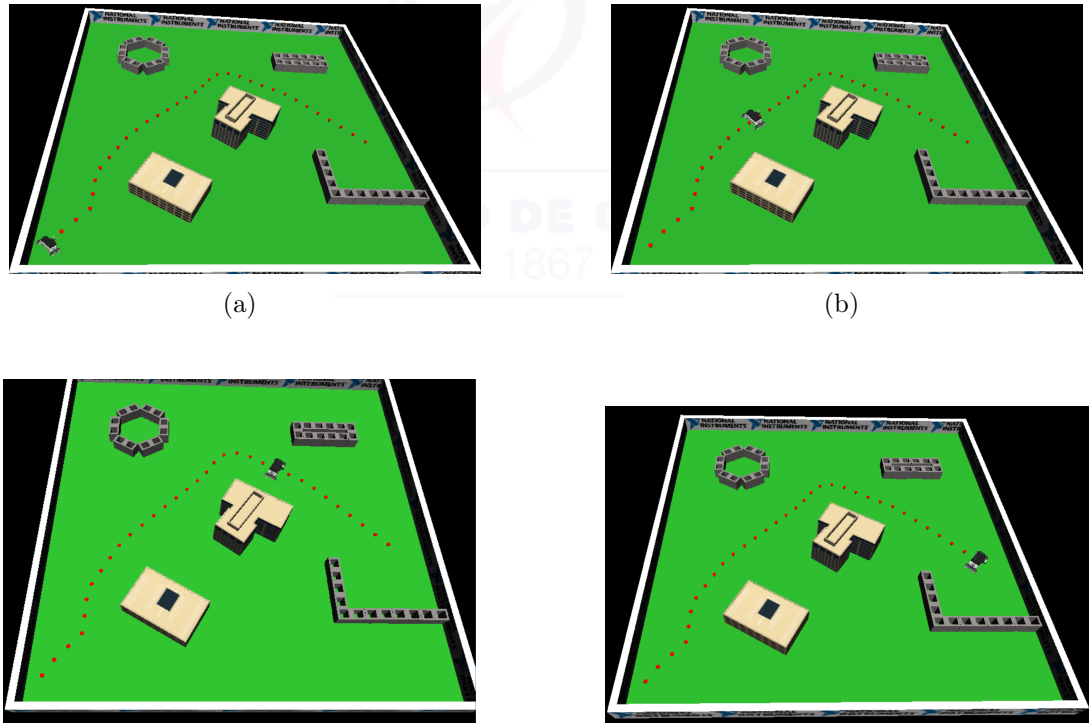
Figura 5.1: Entornos de evaluación del algoritmo FMM^2 implementado en el robot DaNI

5.1. Pruebas de simulación del UGV DaNI en entorno virtual

Las simulaciones de prueba del algoritmo FMM^2 sobre el robot $DaNI$ se realizan primeramente en un entorno 3D virtual con el objetivo de cuidar la integridad del robot durante el ajuste final de parámetros. En contraste, LabVIEW posee un simulador de entornos con propiedades físicas lanzado en LabVIEW Robotics 2011, basado en el Open Dynamics Engine y compatible con las plataformas NI LabVIEW Robotics Starter Kit 1.1 y 2.0. Este simulador puede probar cualquier entrada de sensor y salida de motor, permitiendo ejecutar un test de

algoritmos como mapeo, direccionamiento y planificación de trayectorias. Así, de ésta manera, es posible el desarrollo interactivo, amigable, seguro y óptimo del prototipo final ha implementarse en el robot DaNI.

Para nuestro propósito se ha tomado de referencia los dos entornos ilustrados en la Figura 5.1, cuya elección se cimienta en la cantidad de obstáculos inherentes a cada terreno. Partiendo con la Figura 5.1a, la ruta resultante generada por el algoritmo FMM^2 se ilustra en la Figura 5.2a, y las Figuras 5.2b, 5.2c y 5.2d representan el avance del robot a lo largo de ella; sin embargo, claramente se puede notar un sobredimensionamiento de la ruta, ocasionando una ruta no óptima para nuestro problema.



(a) (b)

(c) (d)

Figura 5.2: Transito del robot DaNI a través del entorno de la Figura 5.1b

Afortunadamente, el sobredimensionamiento solo se debe a la exigencia desmesurada del espaciamiento requerido entre los obstáculos y el robot DaNI. Por lo tanto, puesto que el espaciamiento está directamente relacionada con el mapa de velocidades, la tarea ahora es manipular el mapa de velocidades producido por el algoritmo FMM^2. Bajo este criterio, la nueva ruta, óptima tanto en tiempo como en espacio, se muestra en la Figura 5.3, cuyo resultado cumple con la separación

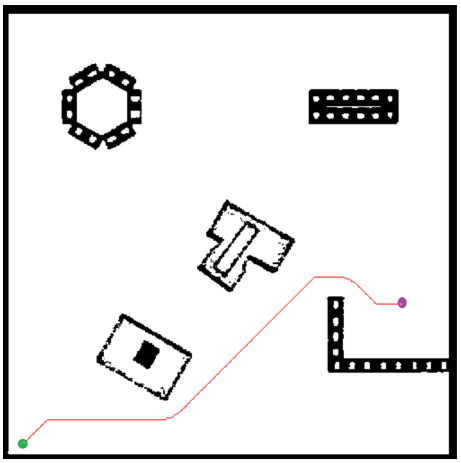
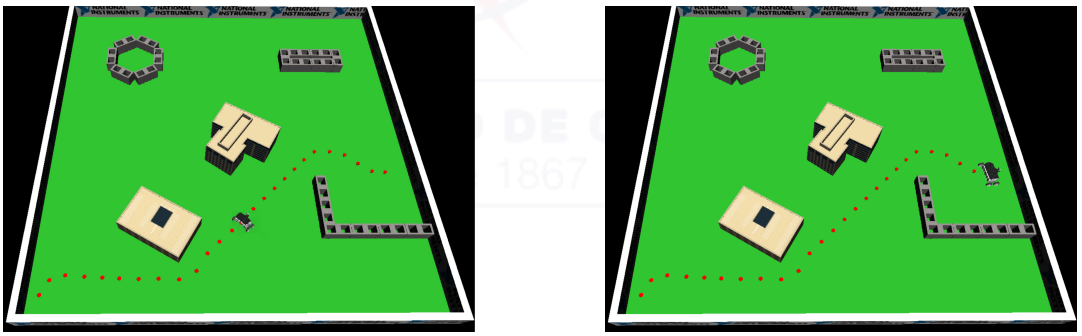


Figura 5.3: Trayectoria no sobrestimada

adecuada para el correcto tránsito del robot DaNI.



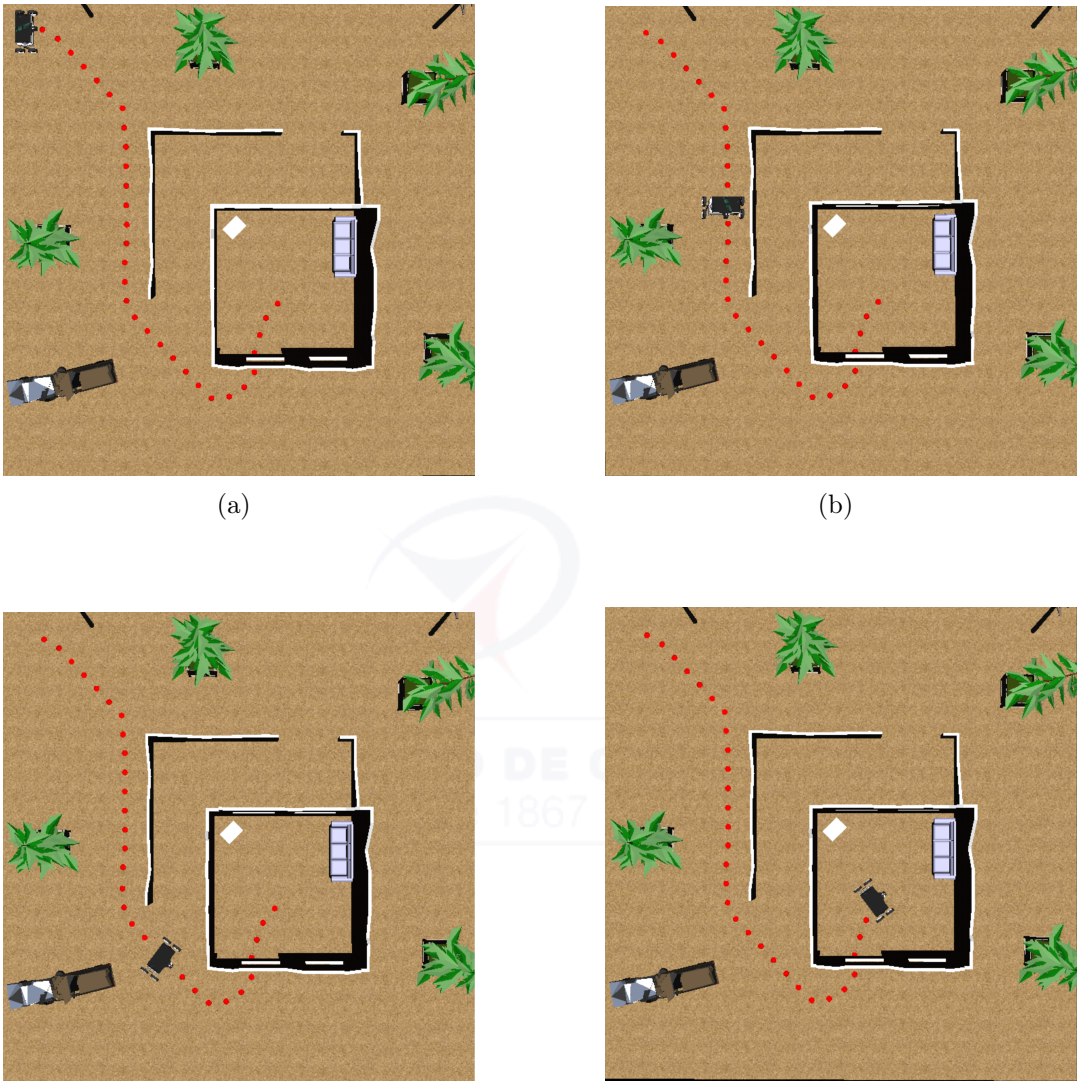
(a) (b)

Figura 5.4: Ruta de Transito del Robot DaNI mejorada

Luego de corroborar la correcta generación de la ruta, se procede a la simulación del robot DaNi en el entorno 3D de LabVIEW. La travesía del robot a lo largo de la ruta hallada está libre de colisiones, ver Figura 5.4a y 5.4b, significando que el dimensionamiento establecido es idóneo para el problema.

Como una medida preventiva adicional, ahora la planificación de la ruta se lleva a cabo en el escenario de la Figura 5.1b, y efectivamente, la ruta producida para dicho escenario también es eficiente, ver Figura 5.5a, 5.5a, 5.5b, 5.5c y 5.5d.

En definitiva, resulta simple notar la gran eficiencia del algoritmo FMM^2 aplicada a la planificación de trayectorias, ofreciendo la posibilidad de lidiar solamente con un único parámetro, la velocidad de expansión de las interfaces F, para amoldarla a las necesidades de cualquier problema.



(a) (b)

(c) (d)

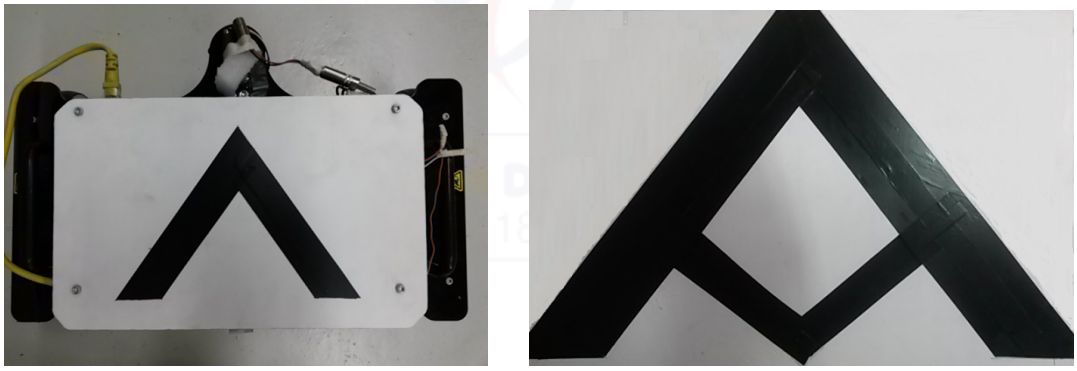
Figura 5.5: Transito del robot DaNI a traves del entorno de la Figura 5.1b

5.2. Pruebas del UGV DaNI en entorno real

Las pruebas realizas en el UGV DaNI se realizan a dos situaciones específicas. La primera prueba consiste en navegar de forma autónoma en un ambiente cerrado. DaNI se desplaza del punto inicio al punto meta siguiendo la ruta óptima. En esta prueba su ruta no será interrumpida por aparición de nuevos obstáculos. La siguiente prueba consiste en navegar de forma autónoma en un ambiente cerrado, donde aparecen nuevos obstáculos en la ruta óptima.

En el entorno de pruebas, se debe tener presente la existencia de limitaciones

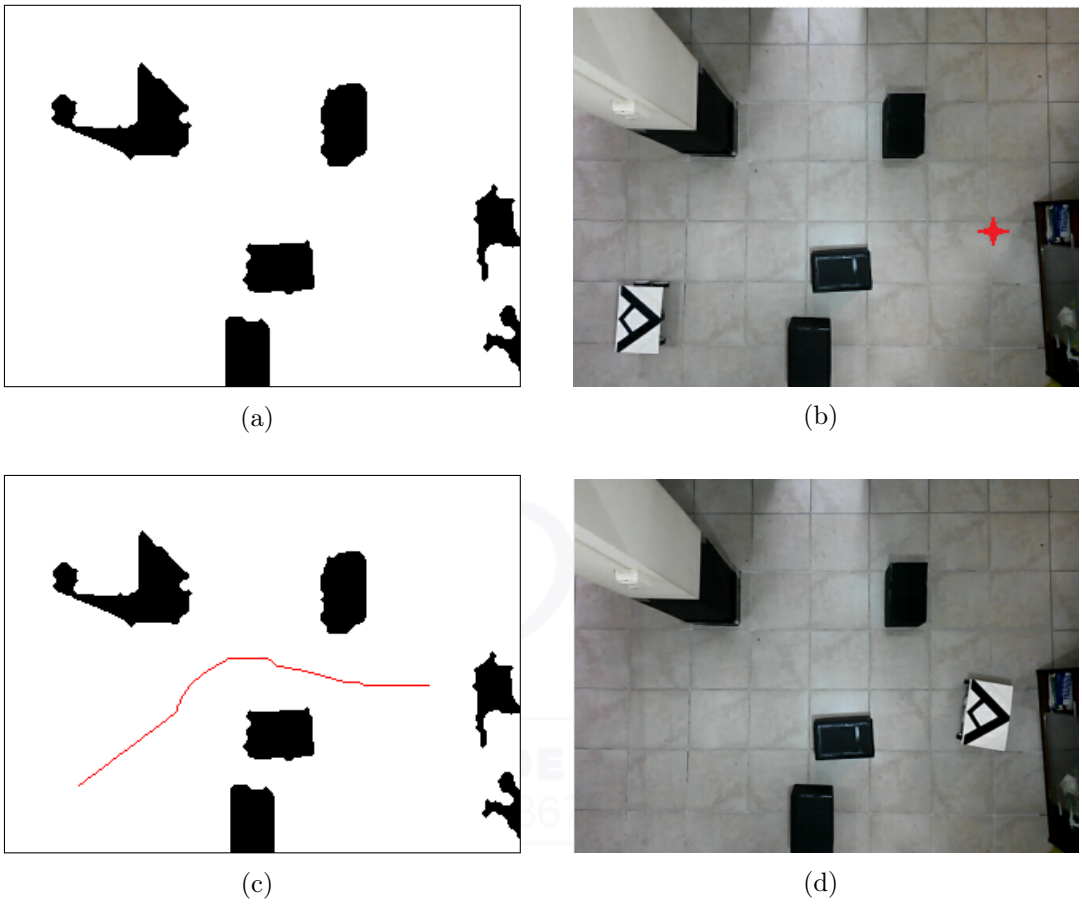
como son la dimensión, que se encuentra restringido a un FOV de 320 cm × 240 cm debido a las características del dispositivo de captura y adquisición de imagen empleado. La altura WD se encuentra limitada por la altura del entorno de prueba y la iluminación es variable de acuerdo a la incidencia de luz natural y del lugar de prueba. Las limitaciones existentes se controlan en mediante la calibración de parámetros como son el control de escala de píxel a cm, controles de umbrales de binarización y fuentes de iluminación artificial si fuese necesario. Para las pruebas realizadas se modificó el símbolo de reconocimiento que se encuentra en la parte superior de DaNI (Figuras 5.6a y 5.6b) siendo en nuevo símbolo algo particular (con respecto al entorno de DaNI) evitando así algunos errores debido a objetos que generaban patrones con cierta similitud al símbolo anterior de DaNI.



(a) Símbolo anterior (b) Símbolo nuevo

Figura 5.6: Símbolo de reconocimiento sobre DaNI

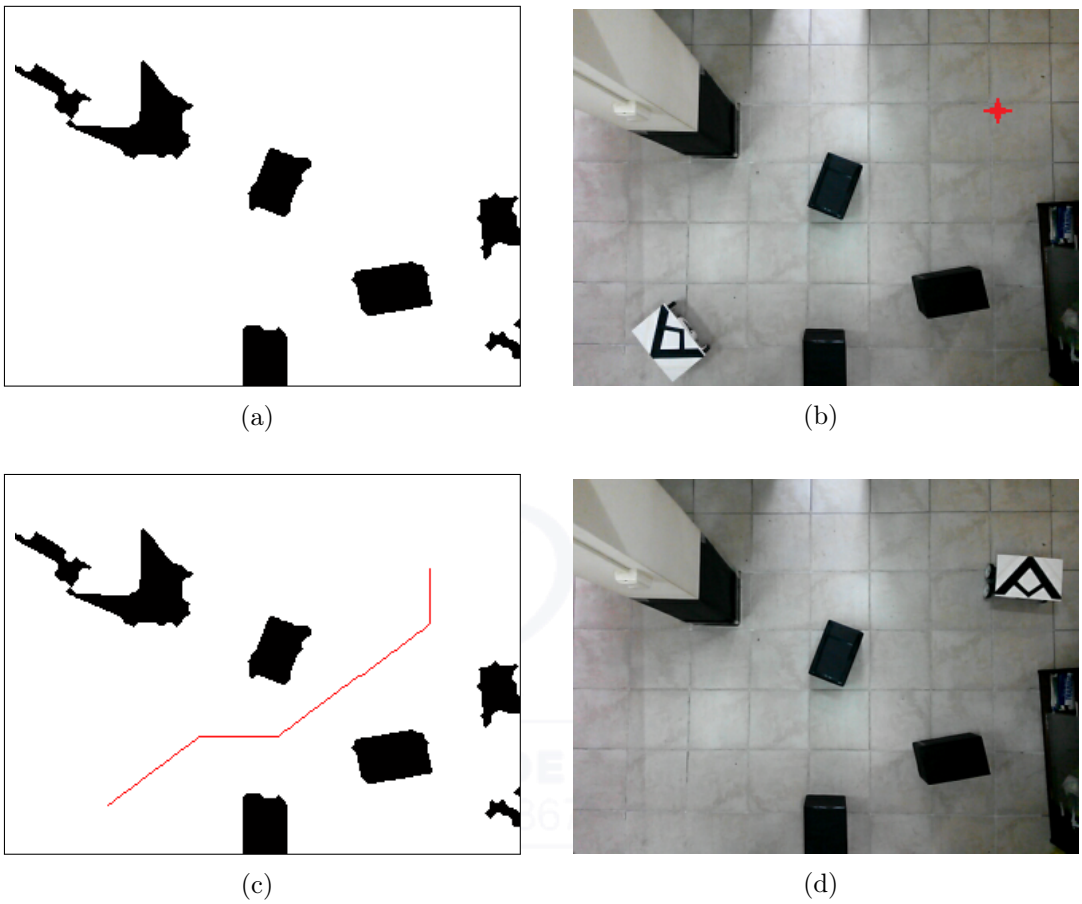
Los resultados de la primera prueba de navegación autónoma de DaNI se observan en las Figuras 5.7 y 5.8 (la prueba se realiza para dos casos), donde se aprecian a) la imagen binaria del entorno luego del DIP, b) el entorno en que se encuentra DaNI y el punto meta, c) la ruta óptima calculada y d) el posicionamiento final de DaNI.



(a) (b)

(c) (d)

Figura 5.7: Desplazamiento del UGV DaNI, sin la presencia de obstáculos en su ruta (prueba 1.a)



(a) (b)

(c) (d)

Figura 5.8: Desplazamiento del UGV DaNI, sin presencia de obstáculos en su ruta (prueba 1.b)

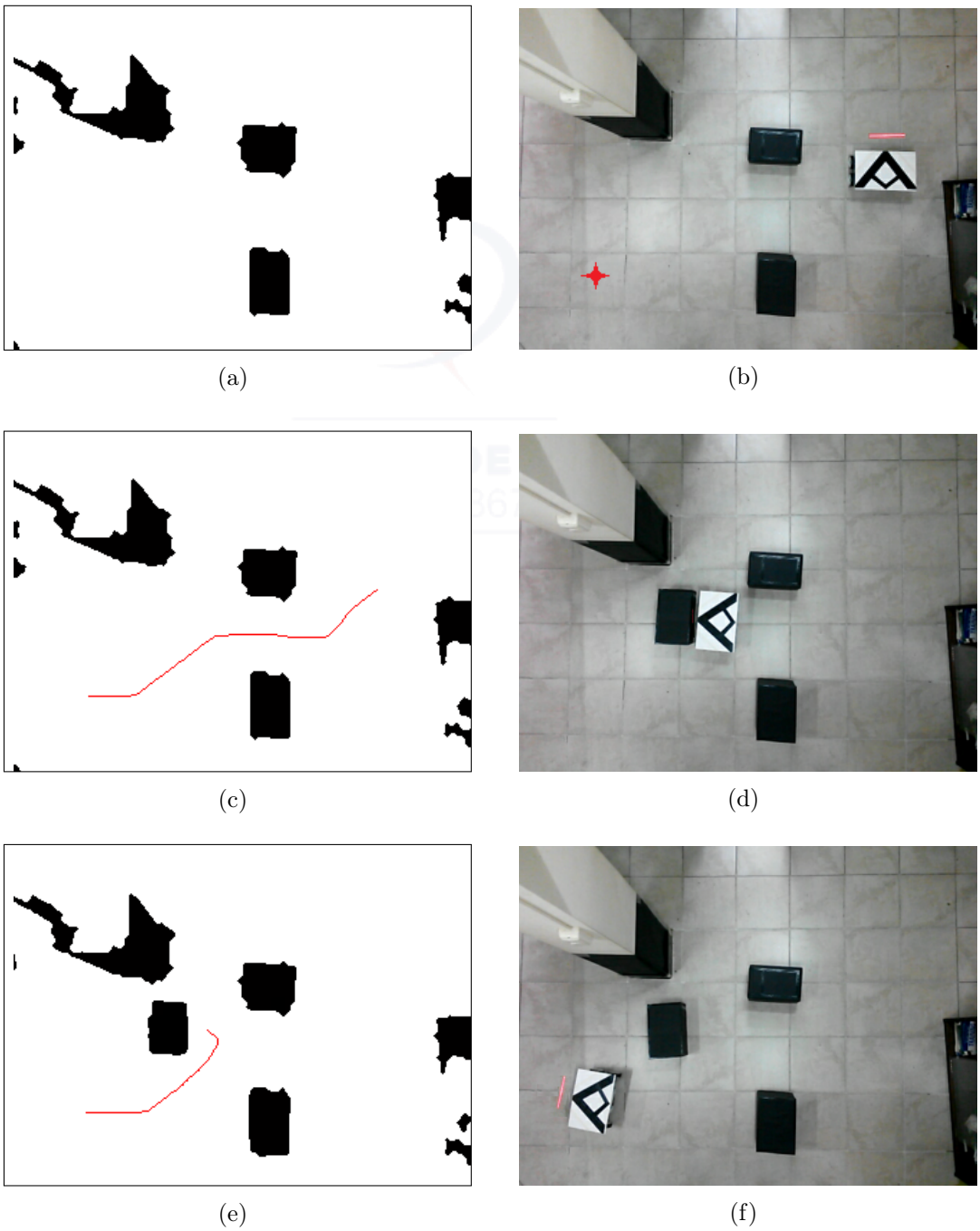
En la Tabla 5.1 se presenta los valores de posición y dirección inicial, y punto meta del UGV DaNI obtenidos en la prueba 1. Los resultados obtenidos permiten finalizar satisfactoriamente la prueba 1.

Tabla 5.1: Desplazamiento del UGV DaNI (prueba 1)

Prueba	**Punto de inicio [píxeles]**	**Dirección [^{o}]**	**Punto meta [píxeles]**
1.a	(41 , 197)	353	(263 , 135)
1.b	(59 , 209)	217.9	(262 , 60)

Los resultados de la segunda prueba de navegación autónoma de DaNI se observan en la Figura 5.9, donde se aprecian a) la imagen binaria del entorno luego del DIP, b) el entorno en que se encuentra DaNI y el punto meta, c) ruta óptima calculada, d) presencia de obstáculo en la ruta, e) ruta óptima nueva y d) ubicación final de DaNI.
En la prueba realizada el objeto bloquea la ruta a DaNI de forma indefinida,

requiriéndose el calculo de una nueva trayectoria. Si el objeto se retira en un tiempo menor a 5 segundos DaNI sigue con su ruta original. La Tabla 5.2 se presenta los valores de posición y dirección inicial, posición y dirección de DaNI al encontrarse con el obstáculo, y punto meta del UGV DaNI obtenidos en la prueba 2.



(a) (b) (c) (d) (e) (f)

Figura 5.9: Desplazamiento del UGV DaNI, con presencia de obstáculos en su trayecto (prueba 2)

Tabla 5.2: Desplazamiento del UGV DaNI (prueba 2)

Punto inicial [píxeles]	**Dirección inicial [^{o}]**	**Nuevo punto inicio [píxeles]**	**Nueva dirección inicial [^{o}]**	**Punto meta [píxeles]**
(255 , 112)	89.53	(137 , 130)	175	(52 , 187)

Finalmente, al haber obtenido resultados favorables en las pruebas realizadas con DaNI se finalizó satisfactoriamente el desarrollo del UGV, involucrando características sensoriales, generación de la ruta óptima y algoritmos de detección y evasión de obstáculos. No obstante, el problema de desplazamiento, introducido por el tiempo de ejecución del reconocimiento del patrón de luz estructurada, intervenía en el desempeño de los motores de DaNI, ocasionando grandes errores longitudinales en su trayectoria. Dicho problema se resolvió utilizando la ejecución en paralelo de los algoritmos involucrados (desplazamiento y detección de obstáculos por medio del reconocimiento del patrón de iluminación estructurada). A su vez, otra alternativa fué realizar la detección de obstáculos en cada segmento de la trayectoria de DaNI, similar a como lo realizan los robots *Spirit Rover* y *Opportunity Rover* [4].

Capítulo 6

Conclusiones

En este capítulo final se resumen las principales ventajas de este trabajo de investigación, resaltando los resultados claves.

Las plataformas autónomas actualmente desenvuelven un papel fundamental en aplicaciones terrestres tales como la expedición de zonas inhóspitas, tareas de predicción e inclusive reconstrucciones de entornos 3D. Sus altos niveles de autonomía y su habilidad de permanecer largos periodos de tiempo sin la intervención del hombre son propiedades muy atractivas.

Las necesidades sensoriales del UGV DaNI son resueltas mediante visión artificial, sensor de ultrasonido e iluminación estructurada, permitiendo conocer el posicionamiento del robot, tener una imagen binaria que representa los objetos presentes en en entorno y detectar la existencia de obstáculos. Los elementos y algoritmos usados para los mismos pueden operar de forma independiente o complementaria permitiendo tener redundancia y/o relevase según sea necesario.

Existen variedad de algoritmos dedicados a la planificación de rutas, unos mejores que otros. Este documento a recopilado los siguientes 5 algoritmos: RRT, $B-RRT$, A^*, FMM, FMM^2. El primer par de algoritmos poseen una convergencia rápida, pero proveen rutas extremadamente irregulares. Contrariamente, el algoritmo A^* y FMM generan rutas menos pronunciadas, pero adyacentes a los obstáculos. A diferencia de los 4 algoritmos anteriores, FMM^2 ofrece rutas regulares y la propiedad de manipular el espaciamiento entre la ruta óptima y los obstáculos.

Este trabajo de titulación ha desarrollado una metodología eficiente y rigurosa basada en el algoritmo FMM^2, para resolver el problema de planificación de la ruta de mínimo tiempo para un vehículo autónomo en un campo estático, es decir, con velocidad dependiente únicamente de la distancia. Dicha solución está basada en la resolución de la ecuación diferencial Eikonal, consecuencia directa de la

formulación de Valor de Frontera. Bajo este criterio, se han realizado simulaciones exitosas en entornos con distintivas propiedades, ilustrando la versatilidad de la aproximación del algoritmo FMM.

Una importante restricción del uso del algoritmo FMM^2 es el costo computacional invertido, ya que el algoritmo es producto de una ejecución por duplicado del algoritmo FMM. El tiempo de procesamiento del algoritmo FMM^2 es aproximadamente 40 segundos, resultando una desventaja en aplicaciones que requieran la selección rápida de rutas de transito óptimas. Por tal motivo, es de vital importancia realizar un escalamiento de la imagen a procesar para asegurar una convergencia más rápida.

FMM^2 ofrece flexibilidad en la generación de rutas, facilitando la manipulación del espaciamiento requerido entre la ruta y los obstáculos en base al área del robot DaNI. No obstante, cuando el espaciamiento entre la ruta y los obstáculos es exagerado, se produce una sobrestimación de la ruta. Por tal motivo, es necesario ser cuidadosos en determinar la distancia adecuada entre la ruta y los obstáculos para determinar la ruta más corta, permitiendo optimizar el consumo de energía del robot DaNI.

Bibliografía

[1] Cristina Urdiales, Antonio Bandera, and Francisco Sandoval. Historia y tendencias actuales de la robótica.

[2] Stanford University. Stanford cart. http://web.stanford.edu/ learnest/-cart.htm.

[3] California Institute of Technology Jet Propulsion Laboratory. Nasa facts. http://www.jpl.nasa.gov/news/fact-sheets/mpf.pdf.

[4] Ramón Silva Ortigoza. Robots móviles: Evolución y estado del arte.

[5] Erik Zamora. Robots autónomos: Navegación. https://ezamorag.wordpress.com/2015/11/08/robots-autonomos-navegacion/.

[6] National Instruments. Ni labview robotics starter kit. http://www.ni.com/datasheet/pdf/en/ds-217.

[7] Raytheon. Raytheon unveils lighter, faster, stronger second generation exoskeleton robotic suit. http://raytheon.mediaroom.com/index.php?item=1652.

[8] Raytheon. Raytheon unveils lighter, faster, stronger second generation exoskeleton robotic suit. http://raytheon.mediaroom.com/index.php?item=1652.

[9] Roe. Uran-6 mine-clearing robot, russia. http://www.army-technology.com/projects/uran-6-mine-clearing-robot/.

[10] Michelle Starr. A bomb-disposal robot designed for darpa has hands created from prosthetics designed for amputees. http://www.darpa.mil/.

[11] LUDEK ZALUD, TOMAS JILEK, and PETRA KOCMANOVA. Robotic autonomous field gamma radiation measurement for environmental protection.

[12] Emilio Gómez González. *Fundamentos de óptica: apuntes de óptica: curso 2010-11*. Universidad de Sevilla, 2010.

[13] Jose Pardeiro Blanco. *ALGORITMOS DE PLANIFICACIÓN DE TRAYECTORIAS BASADOS EN FAST MARCHING SQUARE*. Master, Universidad Carlos III de Madrid, LEGANÉS, MADRID, February 2015.

[14] Tekin Meriçli, Manuela Veloso, and Levent Akin. Improving prehensile mobile manipulation performance through experience reuse. *International Journal of Advanced Robotic Systems*, 12, 2015.

[15] Plamen S Koev. 18.336 numerical methods of applied mathematics ii, spring 2005. 2005.

[16] JA Sethian. Fast marching methods and level set methods for propagating interfaces. *Lecture series-van Kareman Institute for fluid dynamics*, 3:A1–A59, 1998.

[17] James A Sethian. Adaptive fast marching and level set methods for propagating interfaces. *Acta Math. Univ. Comenianae*, 67(1):3–15, 1998.

[18] Edwin Francis Cárdenas Correa et al. *Una Aplicación Computacional De Generación De Trayectorias y Evasión De Obstáculos Para Distintos Robots Manipuladores Con Análisis En Un Sistema Real*. PhD thesis, Universidad Nacional de Colombia.

[19] Allan Chikwanha, Sibonelo Motepe, and Riaan Stopforth. Survey and requirements for search and rescue ground and air vehicles for mining applications. In *Mechatronics and Machine Vision in Practice (M2VIP), 2012 19th International Conference*, pages 105–109. IEEE, 2012.

[20] Jan Mazal, Petr Stodola, and Milan Podhorec. Ugv development with supervised autonomy. In *Mechatronics-Mechatronika (ME), 2014 16th International Conference on*, pages 359–363. IEEE, 2014.

[21] Panagiotis Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, 26(4):1373–1385, 2013.

[22] RN Farah, Raja Lailatul Zuraida, Umairah Shaharum, and N Irwan. Challenging of path planning algorithms for autonomous robot in unknown environment. *Lecture Notes on Information Theory Vol*, 2(2), 2014.

[23] Daniel Jackson. Research and component systems of unmanned ground vehicles: A survey. *University of Northern Iowa*, 11014.

[24] Robert King. Mobile robotics experiments with dani.

[25] José Andrés Somolinos Sánchez. Avances en robótica y visión por computador. volume 38. Univ de Castilla La Mancha, 2002.

[26] FJ Jiménez, JC Moreno, R González, F Rodríguez Díaz, and J Sánchez-Hermosilla. Sistema de visión de apoyo a la navegación de un robot movil en invernaderos.

[27] IMAQ IMAQ. Imaq vision concepts manual. *National Instruments, Austin*, 2004.

[28] Rafael C González, Richard E Woods, Fernando Davue Rodríguez, and Luis Rosso. *Tratamiento digital de imágenes*, volume 3. Addison-Wesley New York, 1996.

[29] José Rafael Lajara Vizcaíno and José Pelegrí Sebastiá. *LabView: entorno gráfico de programación.* Marcombo, 2011.

[30] National Intruments. ¿por qué usar labview? http://www.ni.com/academic/why_labview/esa/.

[31] NI Vision NI Vision. Ni vision for labview user manual. *National Instruments, Austin*, 2011.

[32] National Instruments. Labview robotics and starter kit dani.

[33] Figueroa Olmedo, Junior Rafael, Palaguachi Alcívar, and Alex Angel. *Diseño y elaboración de guías de laboratorio para la Plataforma móvil labview robotics SBRIO Starter Kit de National Instrument.* PhD thesis, SANGOLQUÍ/ESPE/2012, 2012.

[34] I Parallax. Ping))) ultrasonic distance sensor (# 28015).

[35] Johann Borenstein, HR Everett, Liqiang Feng, et al. Where am i? sensors and methods for mobile robot positioning. *University of Michigan*, 119(120):27, 1996.

[36] S LaValle. Planning algorithms, cambridge univ, 2006.

[37] Steven M LaValle. Motion planning. *IEEE Robotics & Automation Magazine*, 18(1):79–89, 2011.

[38] Lydia E Kavraki and Steven M LaValle. Motion planning. In *Springer handbook of robotics*, pages 109–131. Springer, 2008.

[39] John Kitzinger and B Moret. The visibility graph among polygonal obstacles: a comparison of algorithms. *University of Mexico, available online at: http://www. cs. unm. edu/˜ treport/tr/03-05/Kitzingerthesis. pdf*, 2003.

[40] Victor Fernando Muñoz Martinez. *Planificación de Trayectorias para Robots Móviles*. Doctoral, Universidad de Malaga, July 1995. Depósito Legal: MA-1400-97.

[41] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[42] Steven M LaValle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

[43] Xuân-Nam Bui, Xu An-nam Bui, Philippe Souères, Jean-daniel Boissonnat, Jean-Paul Lamont, Jean-paul Laumond, Projet Prisme, et al. The shortest path synthesis for non-holonomic robots moving forwards. In *INRIA, Nice-Sophia-Antipolis, Research Report 2153*. Citeseer, 1993.

[44] Gonzalo Pajares, José Jaime Ruz, Pablo Lanillos, María Guijarro, Jesús Manuel de la Cruz, and Matilde Santos. Generación de trayectorias y toma de decisiones para uavs. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 5(1):83–92, 2008.

[45] Masoud Nosrati, Ronak Karimi, and Hojat Allah Hasanvand. Investigation of the*(star) search algorithms: Characteristics, methods and approaches. *World Applied Programming*, 2(4):251–256, 2012.

[46] Introduction to the fast marching method.

[47] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2383–2388. IEEE, 2002.

[48] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[49] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[50] Edwin Francis Cárdenas Correa et al. *Una Aplicación Computacional De Generación De Trayectorias y Evasión De Obstáculos Para Distintos Robots Manipuladores Con Análisis En Un Sistema Real*. PhD thesis, Universidad Nacional de Colombia.

[51] Matthew Jordan and Alejandro Perez. Optimal bidirectional rapidly-exploring random trees. *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. MIT-CSAIL-TR-2013-021*, 2013.

[52] Sai V Konakalla. A star algorithm. *Indiana State University*, page 4, 2014.

[53] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[54] Ariel Felner. Position paper: Dijkstra's algorithm versus uniform cost search or a case against dijkstra's algorithm. In *Fourth Annual Symposium on Combinatorial Search*, 2011.

[55] V Patel and C Baggar. A survey paper of bellman-ford algorithm and dijkstra algorithm for finding shortest path in gis application. *International Journal of P2P Network Trends and Technology (IJPTT)*, 5, 2014.

[56] Richard E Korf. Linear-space best-first search: Summary of results. In *AAAI*, pages 533–538, 1992.

[57] Disha Sharma and Sanjay Kumar Dubey. Anytime a* algorithm–an extension to a* algorithm. *vol*, 4:1–4, 2013.

[58] N. J. Nilsson P. E. Hart and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[59] Hang Dinh and Hieu Dinh. Inconsistency and accuracy of heuristics with a* search. *arXiv preprint arXiv:1307.2200*, 2013.

[60] Robert Holte, Jonathan Schaeffer, and Ariel Felner. Mechanical generation of admissible heuristics. *Part I: Heuristics*, page 43.

[61] Richard E Korf and Michael Reid. Complexity analysis of admissible heuristic search. In *AAAI/IAAI*, pages 305–310, 1998.

[62] Sri Venkata Tapovan Lolla. *Path planning in time dependent flows using level set methods*. PhD thesis, Massachusetts Institute of Technology, 2012.

[63] Jeff Dicker. Fast marching methods and level set methods: an implementation. 2010.

[64] James A Sethian. Evolution, implementation, and application of level set and fast marching methods for advancing fronts. *Journal of computational physics*, 169(2):503–555, 2001.

[65] James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[66] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[67] S Garrido, D Blanco, L Moreno, M Abderrahim, and ML Munoz. *Sensor-based Global Planning for Mobile Manipulators Navigation using Voronoi Diagram and Fast Marching*. INTECH Open Access Publisher, 2007.

[68] M Sabry Hassouna and Aly A Farag. Multistencils fast marching methods: A highly accurate solution to the eikonal equation on cartesian domains. *IEEE transactions on pattern analysis and machine intelligence*, 29(9):1563–1574, 2007.